University of Alberta

Library Release Form

Name of Author: Paul Shelley

Title of Thesis: New Techniques in Wavelet Image Compression

Degree: Master of Science

Year this Degree Granted: 2001

Let us therefore brace ourselves to our duties, and so bear ourselves that if the British Empire and Commonwealth last for a thousand years, men will still say, This was their finest hour.
Winston Spencer Churchill

Compression is hard
But I have found a new way
I call it Hy-Q

University of Alberta

New Techniques in Wavelet Image Compression

by

Paul Shelley

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

Department of Computing Science

Edmonton, Alberta
Fall 2001

University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **New Techniques in Wavelet Image Compression** submitted by Paul Shelley in partial fulfillment of the requirements for the degree of **Master of Science.**

For my family.

Acknowledgements

I would like to thank my supervisor, Dr. Xiaobo Li, for working with me in the development of Hy-Q, my image compression algorithm. I would also like to thank my co-supervisor, Dr. Bin Han, for helping me with my numerous questions about the mathematical aspects of wavelet transformations.

My family and friends supported me throughout my research, and I owe them a special thank you for making my Masters program an enjoyable one.

This research was supported in part by Postgraduate Scholarships from the Canadian Natural Sciences and Engineering Research Council (PGS A) and from the Alberta Provincial Government (Ralph Steinhauer and iCore).

Abstract

Effective lossy image compression allows an image to be stored in a small amount of space without sacrificing much visual quality. With the trend towards increased multimedia content in applications and mobile communications, research in this area will continue to develop.

This thesis presents two new contributions to image compression research. The first is an algorithm that combines the quantization strategies of two existing methods. The results improve upon those of both existing methods for many images. Even more important than the results is the algorithm itself for combining the quantization strategies, which can be applied to more recent strategies to achieve even better results.

The second contribution is an investigation of a new technique for performing wavelet transforms. The dual frame transform is very similar to biorthogonal wavelet transforms, but employs multiple highpass filters. Several practical problems arising in the implementation of this transform, and their solutions, are discussed.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The focus of this thesis is image compression, a field that attempts to reduce the amount of storage required for digital images. With the increasing use of multimedia in our society, as well as the trend towards wireless communications, it is important to develop new image compression algorithms that will produce higher quality images at lower bitrates than even today's excellent algorithms can achieve.

1.1 A Standard Image Compression Model

The most common approach to compressing images, seen in Figure 1.1 involves three components, as follows:

Figure 1.1: Compression process

- Mapper - this component transforms the input pixels into a more easily compressed representation. Sometimes, this process directly reduces the amount of space required to store an image, but this is not always the case. For instance, the wavelet transform, which will figure prominently in this thesis, transforms the highly correlated spatial information stored in the image pixels into statistically independent frequency coefficients. For most standard wavelet filters, there are the same number of wavelet coefficients as image pixels, but the wavelet coefficients are concentrated around zero, and thus are easier to compress.

- Quantizer - this component is responsible for reducing the number of distinct values used in the representation of the image. Quantization is a lossy, irreversible operation, but can be done without sacrificing much image quality because the very small changes in pixel intensity associated with quantized data do not affect normal visual processing.

- Symbol Encoder - the final component removes any redundancy among the values output by the quantizer. Two of the most popular symbol encoders are the Huffman coder and the arithmetic coder.

The decompression process, seen in Figure 1.2, is simply the opposite of the compression process. The symbol decoder and inverse mapper undo the effects of their encoding counterparts, producing an approximation to the original image. Note that because of the fact that the quantizer is a lossy component, its effects cannot be undone, and thus there is no inverse quantizer in the decompression process.

Figure 1.2: Decompression process

1.2 Current Techniques in Image Compression

The current techniques in image compression can achieve results that are nearly visually lossless at compression ratios up to 50:1. Many of these algorithms build upon a concept introduced by J. Shapiro in [Sha93]. His Embedded Zerotree Wavelet (EZW) algorithm exploited the similarity among the coefficients at corresponding spatial locations of different levels of the wavelet coefficient hierarchy. EZW allowed the most significant coefficients to be coded first, and coded many coefficients that are insignificant to the compression process with just one small symbol. This very powerful idea, which requires very little computation, achieved results that were competitive with, or better than, those of any previous method.

Building on this idea, Said and Perlman introduced Set Partitioning in Hierarchical Trees (SPIHT) in [SP96]. This algorithm improved upon EZW by considering the wavelet coefficients in sets instead of individually. This allowed even more insignificant coefficients to be coded with a single symbol. SPIHT has been a baseline for comparison ever since it was developed. Papers such as [RH99] point out improvements and optimizations over SPIHT, achieving ever more impressive results.

The development of the CREW algorithm in [ZASB95] was part of the motivation behind the soon-to-be-released JPEG-2000 standard. This standard, which is based upon wavelet transforms instead of the Discrete Cosine Transform (DCT) that drives the current JPEG standard, promises to bring many improvements over JPEG, including exact specification of the desired bitrate, and state-of-the-art low bitrate performance ([MGBB00]).

On the mathematical side of compression, work is being done to find a better mapping technique than wavelet transforms. While they have been proven to

achieve excellent results, wavelets have the deficiency that they are effective only on point discontinuities. Images, however, are discontinuous along two-dimensional edges. Work such as that discussed in [CD99] is being done to develop mathematical transforms that can more efficiently represent this type of information than can wavelets.

1.3 Hybrid Methods

The ideal image compression algorithm would be one that would achieve state-of-the-art results on every type of image, no matter what its characteristics. In reality, however, this is very difficult to achieve. Certain features of an image may be difficult for one algorithm to process, while the same features may be ideally suited to another algorithm.

An example of this can be seen by examining SPIHT and one of its successors, ModLVQ, which was introduced in [Kni96]. ModLVQ used the same framework as SPIHT, but replaced SPIHT's scalar quantization with a form of vector quantization. Experimentation proved that detailed images were better handled by ModLVQ, while smooth images were more suited to being compressed with SPIHT.

Images take on many different forms, and it is impossible to classify them as completely detailed or completely smooth. Most images will have regions that have a large amount of detail, and regions with almost no detail. For instance, an outdoors picture may feature a smooth sky at the top, and a detailed forest at the bottom. One approach to handling this type of image would be to mix the two different algorithms - use SPIHT to compress the sky, and ModLVQ to compress the forest. This is the approach that was taken by the new algorithm Hy-Q, introduced in this thesis.

Other examples of this "hybrid" approach abound in the literature. For instance, in [Tha96], Thao introduces an algorithm that mixes the effects of two different mappers. A fractal compression scheme is used to code the edges and smooth regions of the image, while a DCT-based scheme is used to code the detailed regions.

1.4 Frame Transforms

One new mathematical technique that is being developed for transforming images is to use frames. While standard wavelet filters employ one lowpass filter and one highpass filter, frames use multiple highpass filters. This leads to extra data, but the freedom in design associated with having extra highpass filters imparts more desirable theoretical properties to these data.

There are two main types of frames. Tight wavelet frames, which are generated by a single set of functions, have received a large amount of attention in the

literature, both from the theoretical and the practical point of view. Dual wavelet frames, which are generated from a pair of function sets, on the other hand, have received almost no attention from a practical perspective yet. This thesis presents findings from implementing the dual wavelet frame transform.

1.5 Thesis Outline

This thesis is outlined as follows. Chapter 2 presents the new quantization scheme for wavelet image compression, based on SPIHT and ModLVQ. Chapter 3 presents work done on implementing the dual wavelet frame transforms. Chapter 4 summarizes the work done on both projects, and draws some links between them.

Chapter 2

Hy-Q : Combining Quantization Strategies In Wavelet Image Compression

This chapter presents a new contribution to the literature on wavelet image compression. The new algorithm, called Hy-Q (Hybrid Quantization), combines the quantization strategies of two different existing algorithms within the same compression framework.

The work is organized as follows: Section 2.1 gives an overview of the literature in the image compression area, with emphasis on the material that contributed directly to the development of Hy-Q. Section 2.2 describes the process of measuring certain properties of an image in order to decide when to apply each quantization strategy. Section 2.3 discusses the evolution of Hy-Q, starting with early rejected versions of the algorithm, and showing how the lessons learned from these versions gave rise to Hy-Q. This section concludes with an in-depth study of Hy-Q itself. Finally, Section 2.4 presents the compression results achieved by Hy-Q.

2.1 Background

2.1.1 Wavelets

The wavelet transform has become one of the most popular techniques used in the mapping steps of the compression process, seen in Figures 1.1 and 1.2. Wavelets act on an image to change the spatial information contained in the pixels to frequency information which is more easily compressed.

One of the major advantages of the wavelet transform is that it preserves spatial information in addition to telling us about the frequencies present in an image. This is not the case with other transforms such as the Discrete Cosine Transform (DCT) that is found in the JPEG standard. As will be seen later in this chapter, this feature will play an important role by allowing a connection to be made between

wavelet coefficients, and the area in the original image from which they came.

The wavelet transform is composed of two operations. The first is the application of a lowpass filter, which produces the general shape of what the data look like. The second is the application of a highpass filter, which captures the detail left behind by the lowpass filter. Because we are dealing with images, which are discrete, these filters take the form of masks, and the filtering is done with discrete convolution. The following equations describe the forward transform process in one dimension, where C^0 is the original data array:

$$C^1 = L * C^0$$
$$D^1 = H * C^0$$

The standard notation, as used above, is fairly intuitive. L and H derive their names from Lowpass and Highpass, while C and D refer to Coarse and Detail, respectively.

In order to reconstruct the original data, the inverse wavelet transform is applied, as follows:

$$C^0 = L^{-1} * C^1 + H^{-1} * D^1$$

In theory, the wavelet transform is completely reversible. It should be possible to reconstruct the original data exactly, given all of the coarse and detail coefficients. However, in practice, this is not possible, for two reasons. Machine precision plays a role, but more importantly, in the compression process information is lost in the quantization step. Therefore, when an image is reconstructed, the final inverse mapping step produces only an approximation to the original data. If the compression is done effectively, this approximation will be close to the original.

Space management is an important aspect of implementing the wavelet transform. The convolution process produces as many numbers as the original data array (technically there are more, but by using reflection at the boundaries of the array, a process which will be discussed in Section 3.3, the extra numbers can be ignored). For the forward transform, the two convolutions, lowpass and highpass, would result in twice as many numbers as we started with. This would pose difficulties for a compression program. Fortunately, because of redundancy in the coefficients, every second coefficient can be ignored. This process is called downsampling. For the inverse transform, the reverse process, called upsampling, is performed. Figure 2.1 highlights the transform process.

It may be noticed that in Figure 2.1, only one level of the forward transform is performed. It is possible to begin the quantization step after just one application

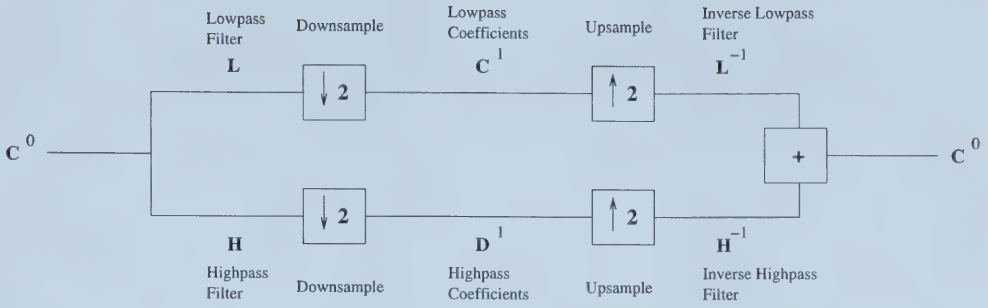


Figure 2.1: The wavelet transform - forward and inverse

of the transform. However, because of a nice property of the wavelet transform, it is desirable to carry the process to multiple levels.

One of the reasons that the wavelet transform is so successful is that it produces many small coefficients, which can be ignored by the quantizer without sacrificing much quality in the reconstructed image. The lowpass filter produces coefficients that are representative of the image at the current spatial location, and thus can be large or small. The highpass filter, on the other hand, produces coefficients that are predominantly small. Because of the mathematical properties that underlie the construction of wavelet filters, areas in the image that contain little variation in pixel intensity will produce coefficients that are close to zero. It is only in regions of large pixel variation, such as edges, that large detail coefficients are found. For the majority of images, the ratio of small detail coefficients to large is quite high. For instance, Figure 2.2 shows the distribution of coefficients from the image House, seen in Figure 2.13. The strong concentration of coefficients with small magnitudes is easily seen.

Because small coefficients are much more easily compressed than large ones, and because large coefficients often result from the application of the lowpass filter, it makes sense to recursively apply the transform process on the lowpass coefficients. Coefficients with larger magnitude are more important to the reconstruction process, as they contribute more to the precision of the reconstructed image than smaller coefficients do. Thus, the efficiency of the compression process can be improved if more detail, and fewer coarse coefficients, are generated.

So far, details of the wavelet transform have been given in only one dimension. For an image, a two-dimensional transform is necessary. Two options are available. Infrequently, a two-dimensional filter is used. The far more popular choice is to use one-dimensional filters twice, in a tensor product. In other words, all of the rows in an image are transformed, and then the resulting columns are transformed. This results in four distinct regions of coefficients, all of which together take up exactly the same amount of space as the original image, due to downsampling. An image

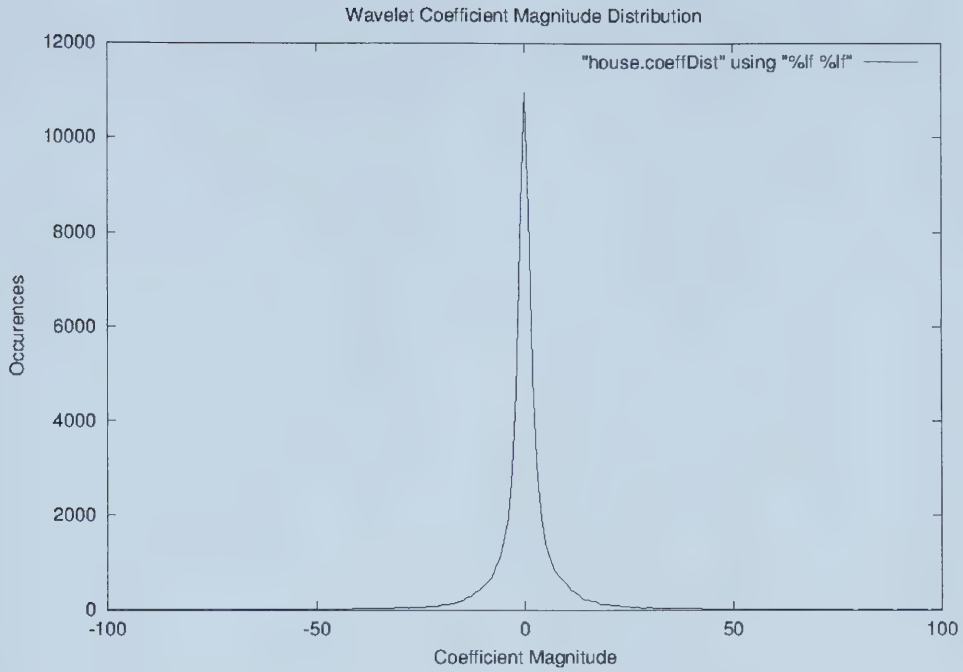


Figure 2.2: Distribution of coefficient magnitudes from the test image House

after one and two levels of transform can be seen in Figure 2.3.

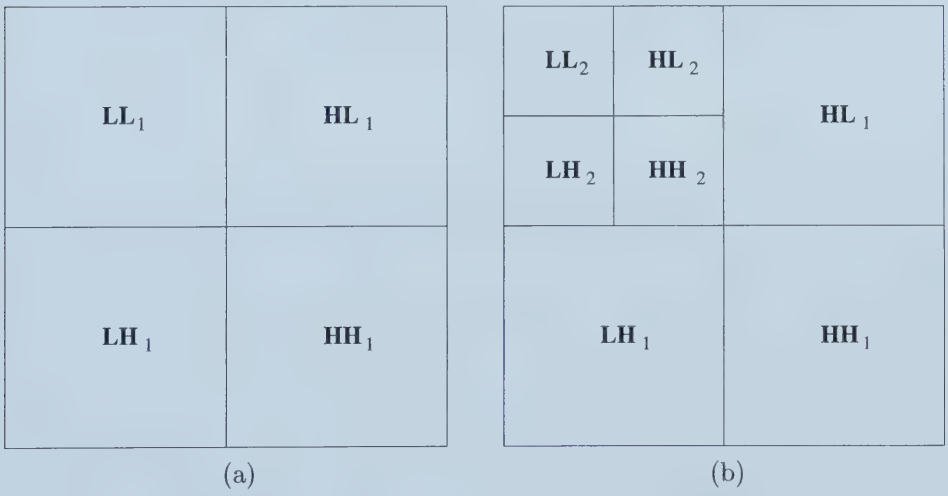


Figure 2.3: Wavelet transform after (a) 1 level (b) 2 levels

Various different names have been given to the different subbands of the wavelet transform hierarchy, and to the different regions within each subband. The convention that will be adopted in this thesis is to label the subbands with the term

Resolution. In each subband, the HL region is Orientation 0 (horizontal detail), the HH region is Orientation 1 (diagonal detail) and the LH region is Orientation 2 (vertical detail). This can be seen in Figure 2.4.

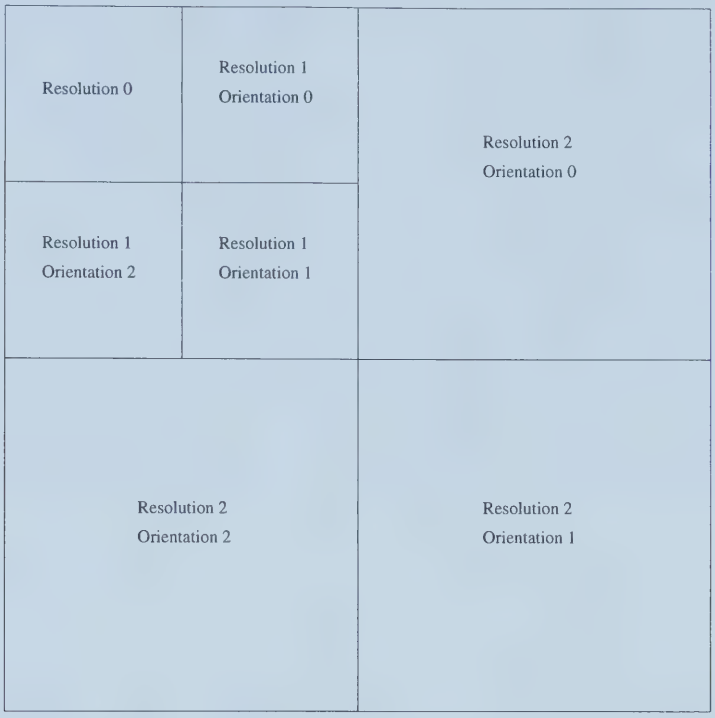


Figure 2.4: Subband labels

2.1.2 EZW

After the wavelet transform has been performed on an image, the next step in the compression process is to quantize the coefficients. Regardless of the quantization strategy chosen, it is of the utmost importance to be able to include the coefficients in the quantization process in decreasing order of magnitude. This is because, as was seen in the previous section, larger coefficients are more significant in the compression, and reduce the error of the reconstructed image more than smaller coefficients do.

In his important paper [Sha93], J. Shapiro introduced a powerful heuristic that allows the wavelet coefficients to be quantized in close to descending order. It is based on the following observations:

- The coefficients in the same orientation across different subbands exhibit similar characteristics. Significant or insignificant coefficients in a higher subband will **likely** be mirrored by similar coefficients in a lower subband that correspond to the same spatial location in the original image.

- In many quantization algorithms that attempt to predict where the significant coefficients are in the wavelet hierarchy, too much of the bit budget is devoted to the map describing the location of the significant coefficients, and not enough to the actual coefficient values.

Shapiro's algorithm, the Embedded Zerotree Wavelet algorithm (EZW), attempts to reduce the size of the "significance map". The main tool used for this is the zerotree. Because of the observation about the self-similarity of the coefficients across subbands, insignificant coefficients at one subband can reasonably be expected to have descendants that are all insignificant as well. Descendants of a coefficient are those coefficients that are at the same orientation of lower levels of the wavelet hierarchy, and at the equivalent spatial location. See Figure 2.5 for a picture of descendants in EZW. This group of coefficients, all of which are insignificant (i.e. equivalent to zero for the purposes of reconstruction), and which form a subtree of the wavelet hierarchy, can be coded efficiently with one symbol, called a zerotree. The other symbols used in the significance map will be discussed later.

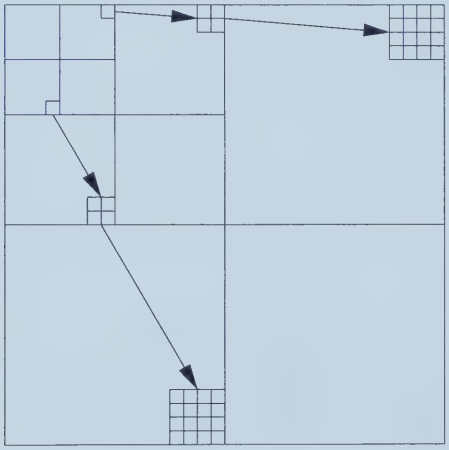


Figure 2.5: Example of Descendant Structure in EZW

An important feature of EZW is that it is an embedded coding scheme. This means that when the bitstream for a particular compression ratio is generated, the bitstream for all higher compression ratios is embedded at the beginning. In effect, the most significant coefficients are "embedded" at the beginning of the bitstream. A good example of this can be seen in the binary representation of real numbers. Given the representation of a number at one level of precision, more bits can be added on to the end to achieve even more precision. The original, lower precision representation is still present in the new representation, right at the beginning.

In order to achieve this embedded bitstream, EZW codes the coefficients one bitplane at a time, from the most significant to the least. In this way, the information

coded at the beginning of the bitstream conveys the most information.

The bitplane coding is achieved by using thresholds that correspond to the value of the current bitplane. It is useless to code a bit from a coefficient that is lower than the current threshold, as it is predictably 0. Therefore, a coefficient x is defined to be *insignificant with respect to a threshold* T if $|x| < T$. This information is used not only to determine when a coefficient should begin to be coded bit by bit, but it also allows us to write out the significance map efficiently.

The following is a list of the symbols used for the significance map:

- ZTR (zerotree root): this symbol is coded for a coefficient that is insignificant with respect to the current threshold T, and all of whose descendants are also insignificant with respect to T.
- IZ (isolated zero): used when a coefficient is insignificant with respect to the current threshold, but has at least one descendant that is significant.
- POS (positive significant): indicates that the current coefficient, which is positive, is significant at the current threshold, and should thus begin to be coded
- NEG (negative significant): same as POS, but for a negative coefficient

In order to manage the coefficients, keeping track of which coefficients should be bitplane coded, and which are still insignificant, EZW keeps two lists of coefficients. The first is called the dominant list, containing all of the coefficients that have not yet been determined to be significant with respect to some threshold. The other list is called the subordinate list, containing all of the coefficients that are being bitplane coded.

EZW makes one important refinement that improves over simple bitplane coding. Consider the example of coding the number that first becomes significant at the threshold $T = 32$. Simple bitplane coding would begin by setting the Most Significant Bit to 1, giving a reconstructed value of 32. However, the "uncertainty interval" for this number at this point is [32..63]. In other words, having received just this one bit, the decoder knows only that the value being coded is somewhere between 32 and 63.

From the point of view of minimizing error, 32 is not the best approximation to the magnitude of the coefficient. Shapiro determined experimentally that the error can be most reduced by setting the reconstructed magnitude to the middle of the uncertainty interval, which, in the above example, is 48. For each successive threshold, a new bit is coded, which means that the uncertainty interval can be cut in half. In each case, the reconstructed value of the coefficient is set to be the middle of the new uncertainty interval.

From an implementation point of view, each entry on the subordinate list keeps the following two values:

- *actual_magnitude* - the true value of the coefficient
- *reconstructed_magnitude* - the value at the middle of the current uncertainty interval. Once the encoding has been completed, this is the value that will be used by the decoder when the inverse wavelet transform is begun.

Here is a high-level overview of the EZW algorithm:

1. Read in the image, and determine the desired compression ratio, which is used to calculate the bit budget.
2. Perform the wavelet transform on the image.
3. Calculate the initial threshold T. In order to begin coding at the most significant bitplane of all of the wavelet coefficients, this value is set to be largest power of 2 that is smaller than or equal to the largest wavelet coefficient. This is the largest possible threshold with respect to which there are guaranteed to be significant coefficients.
4. Initialize the dominant and subordinate lists. Since at the beginning, no coefficients have yet been found to be significant, all of them are placed on the dominant list. The order in which they are placed on the list is important. In order to code zerotrees from the highest possible root coefficient, the coefficients are added to the dominant list in descending order of subband. In addition, a prescribed ordering is used in each subband. This is done to avoid having to waste bits on conveying the ordering to the decoder. The ordering used in EZW is illustrated in Figure 2.6.
5. While ($T > 0$ AND the bit budget has not yet been exhausted):
    - Perform dominant pass. In this pass, the dominant list is scanned in the order in which the coefficients were initially put on the list. For each coefficient, the relevant symbol (ZTR, IZ, POS or NEG) is output. Significant coefficients are removed from the dominant list and placed on the subordinate list. The actual_magnitude is set to be the absolute value of the coefficient, and the reconstructed_magnitude is set to be the middle of the current uncertainty interval. This is midway between the current and the previous threshold, i.e. $(T + 2T)/2$.
    - Perform subordinate pass. In this pass, all of the coefficients on the subordinate list are refined. This involves updating the reconstructed_magnitude value (the actual_magnitude value does not change). If the actual_magnitude is lower than the previous reconstructed_magnitude, the new uncertainty

interval is the lower half of the previous one, and a 0 is output. Otherwise, it is the upper half of the previous interval, and a 1 is output. As discussed above, the new reconstructed_magnitude is the middle of the new uncertainty interval.

- $T \leftarrow T/2$

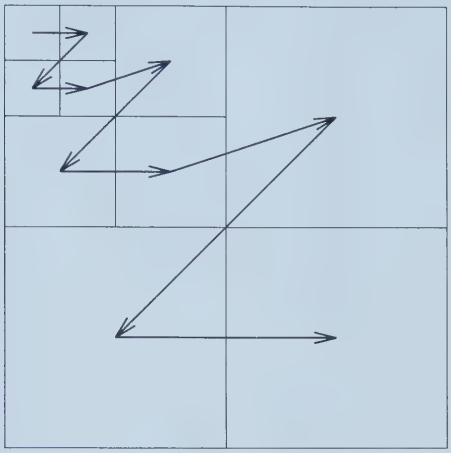


Figure 2.6: Order for placing coefficients on the dominant list in EZW

An example would most likely clarify the concepts in EZW. The following will demonstrate the first dominant and subordinate passes when performed on the simple coefficient hierarchy displayed in Table 2.1.

92	123	87	43	14	-3	18	7
31	-98	-63	12	65	17	-12	2
18	-72	43	18	18	12	-8	4
-3	14	-85	9	5	2	16	0
0	7	-6	14	31	6	-4	12
6	-11	23	2	1	8	-7	11
3	-7	-4	0	-10	0	5	-9
17	12	5	1	6	1	16	2

Table 2.1: Sample 3-level wavelet hierarchy from an 8x8 image

Table 2.2 shows the processing that occurs in the dominant pass. Since the largest coefficient in the hierarchy is 123, the first threshold is selected to be 64. The following comments reference the table:

(1) This coefficient has a magnitude of 92, which is greater than the current threshold of 64. It is positive, and so a POS symbol is output. Its reconstruction magnitude is 96, since that is the middle of the uncertainty interval [64..128), which

Comment	Subband	Coefficient Value	Symbol	Reconstruction Value
(1)	LL3	92	POS	96
	HL3	123	POS	96
(2)	LH3	31	IZ	0
	HH3	-98	NEG	-96
	HL2	87	POS	96
(3)	HL2	43	ZTR	0
	HL2	-63	ZTR	0
	HL2	12	ZTR	0
	LH2	18	ZTR	0
(4)	LH2	-72	NEG	-96
	LH2	-3	ZTR	0
	LH2	14	ZTR	0
	HH2	43	ZTR	0
	HH2	18	ZTR	0
	HH2	-85	NEG	-96
	HH2	9	ZTR	0
	HL1	14	Z	0
	HL1	-3	Z	0
	HL1	65	POS	96
	HL1	17	Z	0
	LH1	-6	Z	0
	LH1	14	Z	0
	LH1	23	Z	0
	LH1	2	Z	0
	HH1	-10	Z	0
	HH1	0	Z	0
	HH1	6	Z	0
	HH1	1	Z	0

Table 2.2: Processing from dominant pass (symbol Z represents a zero with no children)

is the best guess that the decoder would be able to make if the coding were to stop here.

(2) This coefficient, 31, is smaller than the current threshold, and is thus insignificant. However, one of its children in subband HL2 (-72) is significant, and thus it is not a zerotree root. Instead, an IZ symbol is output.

(3) The magnitude 43 is insignificant with respect to the current threshold, and all of its descendants are also insignificant. Thus, a ZTR symbol is output. Note that no symbol needs to be generated for any of the children of this coefficient, as they belong to the zerotree, and thus the decoder can determine that they are insignificant without further information.

(4) -72 is significant, and thus a NEG symbol is output. Since this coefficient

will be removed from the dominant list and have its actual value recorded in the actual_magnitude field of the subordinate list entry, its value in the coefficient hierarchy is set to 0. This will allow its parent, 31, to be coded with a ZTR symbol in the next dominant pass, when the threshold will be 32.

Table 2.3 shows the processing that occurs in the subordinate pass of this example.

actual_magnitude	Symbol	reconstructed_magnitude
92	0	80
123	1	112
98	1	112
87	0	80
72	0	80
85	0	80
65	0	80

Table 2.3: Processing from subordinate pass

The first entry has magnitude 92. This is below the original reconstruction value from Table 2.2, so the new uncertainty interval is the lower half of the previous one, i.e. [64..96). The new reconstructed_magnitude is the middle of the interval, namely 80. The next entry has magnitude 123, which places it in the upper half of the previous uncertainty interval. Its new reconstructed_value is 112, in the middle of [96..128). The other entries are handled similarly.

One final refinement performed by EZW is to reorder the coefficients on the subordinate list after each subordinate pass. This is done in order to ensure that the largest coefficients appear at the front of the list, and are thus first to be refined in each subordinate pass. After the subordinate pass in our example, the coefficients will be in the following order: (123, 98, 92, 87, 72, 85, 65). Note that 85 is not moved in front of 72. This is because from the decoder's point of view, these values are the same (note their identical reconstructed_magnitude), and thus the original order is preserved.

It should be noted that all of the symbols generated by EZW are additionally coded by an adaptive arithmetic coder, to further increase the efficiency of the compression.

The decoding process is straightforward to describe. Due to the elegance of the EZW algorithm, decoding is the same as encoding, with the simple change of the word "output" to "input". This feature also allows bits to be saved from not needing to transmit the location of each coefficient. Since the decoder follows the same steps as the encoder, inputting from the output file instead of writing to it, it implicitly knows which coefficients are being dealt with at any given moment.

EZW is an effective algorithm for compression. It is impressively efficient and it generates an embedded bitstream. Furthermore, its compression results at least matched those of any other compression method existing at the time of the publication of [Sha93]. EZW was used as the basis for many comparisons by later compression schemes, and, as we will see in the next three examples, was the foundation for some of those schemes.

2.1.3 SPIHT

One paper that is referenced by many others, either as a basis for comparison or as the basis for a new algorithm, is [SP96]. In it, Said and Perlman introduce the Set Partitioning in Hierarchical Trees, or SPIHT, algorithm. SPIHT builds directly on EZW, but makes several improvements.

The most important aspect of this algorithm is that instead of treating the coefficients individually, as EZW does, SPIHT first groups them into sets of four coefficients, in 2×2 blocks. The same parent-child relationships that were present in EZW are present in SPIHT. Because of the new set groupings, each coefficient, except those in the lowest subband, has a 2×2 set of four children. The only exceptions are the root coefficients, which were also exceptions in EZW. In SPIHT, only three coefficients in a 2×2 root node have children, as indicated in Figure 2.7.

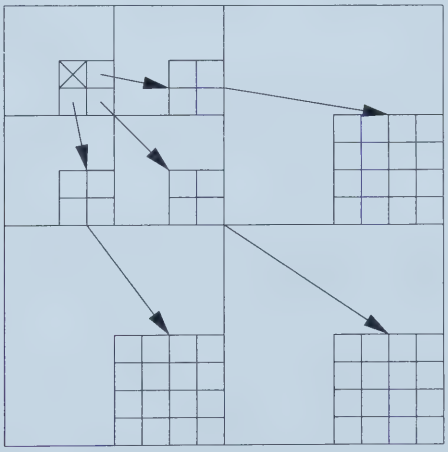


Figure 2.7: Descendants of the root nodes in SPIHT

The major benefit to be gained by grouping coefficients together into sets is that many more insignificant coefficients can be coded with one symbol. Extending the EZW concept of insignificance with respect to a threshold to SPIHT is quite simple. We say that a set S is insignificant with respect to a threshold T if all of the coefficients in S are smaller than T. The thresholds in SPIHT are always powers of 2. Written as a function, and assuming that $T = 2^n$, the significance test looks as

follows:

$$S_n(S) = \begin{cases} 1 & max\{S\} \leq 2^n \\ 0 & otherwise \end{cases} \tag{2.1}$$

SPIHT distinguishes several different types of sets. These are listed in Table 2.4. Individual coefficients are always referred to by their coefficients (i, j).

Set Symbol	Meaning
$D(i,j)$	all descendants of (i,j), at any level
$O(i,j)$	only the direct descendants of (i,j)
$L(i,j)$	$D(i,j) - O(i,j)$
H	all root coefficients

Table 2.4: Sets used in SPIHT

In EZW, coefficients were maintained on two separate lists, depending on their significance. A similar idea is used in SPIHT, but there is an additional list due to the set structure. The following are the lists maintained by SPIHT:

- **LIS** - the list of insignificant sets. These are all of the 2×2 blocks in which no coefficients have been found to be significant at some threshold. Sets of type $D(i,j)$ are referred to by Said and Perlman as type A sets, while sets of type $L(i,j)$ are type B sets.
- **LIP** - the list of insignificant pixels. These are the individual coefficients that belong to a set in which a significant coefficient has been found, but which are not themselves significant.
- **LSP** - the list of significant pixels. These are the individual coefficients that have been found to be significant at a previous threshold.

At the beginning of the algorithm, the root nodes are placed on the LIS. There is a Sorting Pass in SPIHT, in which descendants of sets already on the LIS are considered until all sets containing coefficients significant at the current threshold have been detected. Those sets that do not contain significant coefficients are added to the LIS. The sets that do contain significant coefficients, however, demonstrate the "partitioning" aspect of SPIHT. The set is partitioned into those coefficients that are significant, which are added to the LSP, and those that are not, which are added to the LIP. The Sorting Pass also examines the coefficients on the LIP, moving the newly significant ones to the LSP. In the Refinement Pass, coefficients on the LSP are refined to an additional degree of precision.

The following is a more technical overview of the SPIHT algorithm, as originally written by Said and Perlman in [SP96].

1. **Initialization:** output $\lfloor n = \log_2(max\{\text{coefficients}\}\rfloor$; set the LSP as an empty list and add the coordinates $(i, j) \in H$ to both the LIP and the LIS as type A entries.

2. **Sorting Pass:** (Similar to EZW dominant pass)

   (a) for each entry (i, j) in the LIP do:

      i. output $S_n(i, j)$;

      ii. If $S_n(i, j) = 1$ then move (i, j) to the LSP and output the sign of $c_{i,j}$

   (b) for each entry (i, j) in the LIS do:

      i. if the entry is of type A then

         - output $S_n(D(i, j))$;
         - if $S_n(D(i, j)) = 1$ then
           - for each $(k, l) \in O(i, j)$ do:
             * output $S_n(k, l)$;
             * if $S_n(k, l) = 1$ then add (k, l) to the LSP and output the sign of $c_{k,l}$;
             * if $S_n(k, l) = 0$ then add (k, l) to the end of the LIP;
           - if $L(i, j) \neq \emptyset$ then move (i, j) to the end of the LIS, as an entry of type B;

      ii. if the entry is of type B then

         - output $S_n(L(i, j))$;
         - if $S_n(L(i, j)) = 1$ then
           - add each $(k, l) \in O(i, j)$ to the end of the LIS as an entry of type A;
           - remove (i, j) from the LIS;

3. **Refinement pass:** (similar to EZW subordinate pass)
   For each entry (i, j) in the LSP, **except** those included in the last sorting pass, output the n-th most significant bit of $|c_{i,j}|$;

4. **Quantization-step update:** decrement n by 1 and go to step 2.

Another important improvement of SPIHT over EZW comes in the way in which the coefficients are refined. Recall from Section 2.1.2 that in the Subordinate Pass, **all** coefficients on the subordinate list are refined, which reduces their uncertainty intervals to half their previous length. In its Refinement Pass, SPIHT, on the other hand, does not refine those coefficients that were just added to the LSP. This potentially sacrifices some precision in the quantization of the coefficients just added to the LSP, but it allows for more coefficients to be added to the LSP in

the next Sorting Pass. That this is better can be seen by the fact that the main objective measure of image quality is the Mean **Squared** Error, and thus having more coefficients with moderate reconstruction error is better than having a few coefficients with small error and many coefficients with large error.

The following is an example of the Sorting Pass in SPIHT. It is based on the same simple sample coefficient hierarchy as was used for EZW, with one important change. Only two levels of wavelet transform can be allowed, since the root node must have at least size 2×2 to allow for the size of one SPIHT node. Thus, we will assume that the same grid of numbers arose after two levels of wavelet transform on some original image as arose in the previous example. The new hierarchy can be seen in Table 2.5.

92	123	87	43	14	-3	18	7
31	-98	-63	12	65	17	-12	2
18	-72	43	18	18	12	-8	4
-3	14	-85	9	5	2	16	0
0	7	-6	14	31	6	-4	12
6	-11	23	2	1	8	-7	11
3	-7	-4	0	-10	0	5	-9
17	12	5	1	6	1	16	2

Table 2.5: Sample 2-level wavelet hierarchy from an 8x8 image

The corresponding Refinement Pass will not be given, since, as was explained above, those coefficients that were added to the LSP in this Sorting Pass will not be refined until after the next Sorting Pass. As in EZW, the initial threshold that SPIHT uses for this example is 64.

The following comments reference Table 2.6:

(1) 92 is significant, resulting in the coding of a 1, and it is positive, which results in the coding of a 0. This is part of Step 2(a) in the SPIHT algorithm description, as 92 is initially placed on the LIP.

(2) 31 is insignificant at the current threshold, and thus a 0 is coded.

(3) -98 is significant and negative, each of which results in the coding of a 1.

(4) Now in Step 2(b), the coding of a 1 indicates that there is a significant descendant of 123 at the current threshold.

(5) At the end of the step begun in comment (4), 123 is added to the end of the LIS as a type B set. In this line, the coding of a 1 indicates that there is a significant descendant in $L(123)$.

This Sorting Pass requires 37 bits. The corresponding Dominant Pass in EZW requires 56 bits. This is before arithmetic coding, which is also performed by SPIHT. In addition, the decoding phase of SPIHT, as in EZW, is performed by substituting the word "input" for "output" in the encoding algorithm description.

Comment	Subband	Coefficient Value(s)	Symbol	Reconstruction Value
(1)	LL2	92	1,0	96
	LL2	123	1,0	96
(2)	LL2	31	0	0
(3)	LL2	-98	1,1	-96
(4)	LL2	123	1	N/A
	HL2	87	1,0	96
	HL2	43	0	0
	HL2	-63	0	0
	HL2	12	0	0
	LL2	31	1	N/A
	LH2	18	0	0
	LH2	-72	1,1	-96
	LH2	-3	0	0
	LH2	14	0	0
	LL2	-98	1	N/A
	HH2	43	0	0
	HH2	18	0	0
	HH2	-85	1,1	-96
	HH2	9	0	0
(5)	LL2	123	1	N/A
	LL2	31	0	N/A
	LL2	-98	0	N/A
	HL2	87	1	N/A
	HL1	14	0	0
	HL1	-3	0	0
	HL1	65	1,0	96
	HL1	17	0	0
	LL2	43	0	N/A
	LL2	-63	0	N/A
	LL2	12	0	N/A

Table 2.6: Processing from SPIHT Sorting Pass

SPIHT shares the impressive efficiency of EZW, but improves upon it in terms of compression results. The PSNR of images compressed with SPIHT is usually in the range of 0.5 to 2.0 dB higher than the results produced by EZW.

2.1.4 ModLVQ

In [Kni96], J. Knipe investigated the effects of replacing the scalar quantization used in SPIHT with vector quantization. In general, this strategy involves grouping coefficients together into vectors, quantizing them by matching them up with specially designed vectors from a codebook. The new algorithm is called ModLVQ.

The key issue in designing a vector quantization scheme is choosing the vectors

in the codebook. One approach, used by Averbuch et al in [ALI96], is to perform extensive training prior to running the algorithm, in order to learn which vectors tend to work best. This approach was rejected for ModLVQ, for the following main reasons, among others:

- The training process needs to be long and extensive in order to arrive at a set of useful vectors.
- When the vector matching is done in the quantization step, the entire codebook needs to be searched, which, when there is no regular pattern to the vectors, can take an unacceptably long time.
- It is infeasible to arrive at a codebook containing **all** the vectors that will be needed in the quantization process. It may happen that the vectors arrived at in the training set leave large "holes" in the n-dimensional vector space that would lead to poor matchings of some vectors.

The alternative used in ModLVQ is a strategy called lattice vector quantization. In life, the word lattice refers to "a framework of crossed laths or bars with spaces between, used as a screen or fence, or a structure resembling this" (Oxford English Dictionary). This image is a useful one in picturing what a lattice of vectors might look like. Consider the following example, shown in Figure 2.8, which shows the hexagonal lattice that results from solving the following problem in two dimensions: *how densely can n-dimensional spheres by packed together?*.

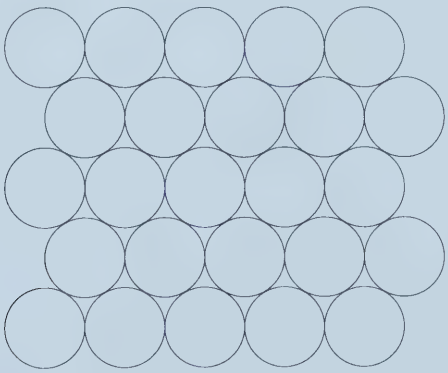


Figure 2.8: A 2-dimensional hexagonal lattice

Since the vectors in a lattice are regularly spaced, the number of large "holes" in the n-dimensional vector space, and thus the possibility of a poor vector match, should be reduced significantly.

Because of the regular coefficient descendant structure in the wavelet hierarchy that is exploited by SPIHT, it is necessary that the groups of coefficients be square.

For ModLVQ, the coefficients are grouped into 4×4 blocks. Each 4×4 node, except for the root node, has four descendants, exactly as in EZW and SPIHT. This can be seen in Figure 2.9.

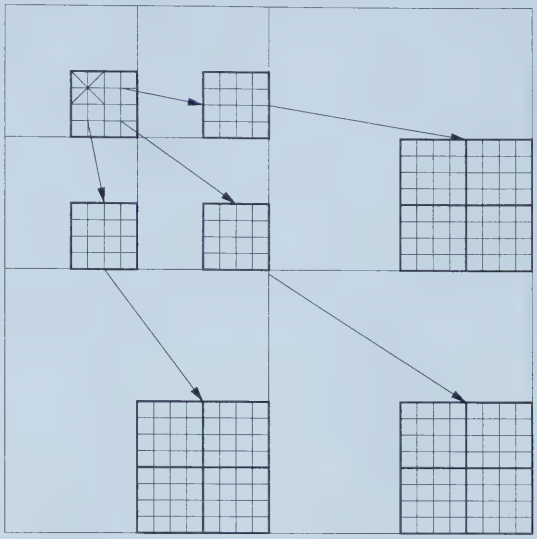


Figure 2.9: Descendant structure in ModLVQ

One major difference between ModLVQ and SPIHT is that in the latter algorithm the threshold is halved after each Refinement Pass. It was found in [SdSG94], a paper that contributed much to the foundation of ModLVQ, that multiplying the threshold by a real constant $a < 1$ gave better results. For ModLVQ, the optimal value of a was determined to be 0.65.

Since the coefficients are no longer being quantized individually, the definition of significance needs to be revised for ModLVQ. It is now the length of the vector that determines whether or not it is significant at a given threshold.

Once a vector is found to be significant, it is represented by the codebook vector that matches it most closely in shape. One issue that needed to be resolved in ModLVQ was how to scale the length of the vectors in the codebook in order to minimize the residual. One approach is to output the most appropriate scaling value. In practice, however, this would have been inefficient, as large numbers of bits would be required for these values. ModLVQ's approach is to use the current threshold as the scaling factor. The residual vector is returned to the list of significant vectors, and matched at the next threshold at which it is significant. The output of matching a vector is the index into the codebook of the matching codevector. In ModLVQ's codebook, there are 8,192 codevectors.

Table 2.7 shows an example of a vector matching performed by ModLVQ during the compression of the test image House, seen in Figure 2.13. The codebook vector

used to match the 16-dimensional coefficient vector has eight non-zero components (eight is the most common number of non-zero components among the vectors used by ModLVQ).

Notice that the five largest wavelet coefficients among the sixteen (231.1840, 173.4099, 147.6488, 145.1225 and 126.9975) are matched with non-zero components, while the next two largest coefficients (96.6224 and 75.0828) are not. Ideally, the eight non-zero vector components should be paired with the eight largest coefficients. However, the vector matching that description was not in ModLVQ's codebook. The residual vector is written back into the coefficient memory, and matched as a new 16-dimensional coefficient vector in the next pass of the algorithm.

Coefficient	Codebook Vector	Vector * Threshold (329.00)	Residual
59.7981	0.0000	0.0000	59.7981
126.9975	0.3536	116.3191	10.6785
145.1225	0.3536	116.3191	28.8034
2.6547	0.0000	0.0000	2.6547
75.0828	0.0000	0.0000	75.0828
173.4099	0.3536	116.3191	57.0908
147.6488	0.3536	116.3191	31.3298
6.3931	0.0000	0.0000	6.3931
231.1840	0.3536	116.3191	114.8649
96.6224	0.0000	0.0000	96.6224
-2.4570	0.0000	0.0000	-2.4570
65.6978	0.3536	116.3191	-50.6213
-11.4251	-0.3536	-116.3191	104.8939
-46.1187	0.0000	0.0000	-46.1187
-37.5706	0.0000	0.0000	-37.5706
56.5250	0.3536	116.3191	-59.7940

Table 2.7: Codebook Vector Matching Example In ModLVQ

This initial approach, however, was found to be imperfect. The reason is that even when a vector is significant at a given threshold, the residual resulting from matching it at the given threshold might be larger than matching it at a later threshold. Thus, new terms, called *optimally significant* and *optimally insignificant* are introduced. Even if a vector's length exceeds the current threshold, it may not be matched in the current pass. This requires one additional bit to be coded for each vector at each threshold after it has been found to be significant, to indicate whether or not it will be coded at the current threshold.

ModLVQ maintains the following two lists of vectors during its processing:

- **LIS** - the list of insignificant sets. This is the list of all the vectors that have not yet been found to be significant at some threshold.

- **LV** - the list of potentially significant vectors. This includes vectors whose initial lengths exceeded one of the thresholds, and the residuals remaining from various iterations of matching the vectors during previous passes.

The following is an overview of the ModLVQ algorithm:

1. **Initialization:** output $T = a * max\{coefficient\_vector\_norms\}$; add the coordinates of the vectors $(i, j) \in H$ to both the LV and the LIS as type A entries.

2. **Sorting Pass:**

   (a) for each entry (i, j) in the LV do:

      i. output $S_T(i, j)$;

      ii. If $S_T(i, j) = 1$ then code the codevector index and put the residual vector back into the coefficient array;

   (b) for each entry (i, j) in the LIS do:

      i. if the entry is of type A then

         - output $S_T(D(i, j))$;
         - if $S_T(D(i, j)) = 1$ then
           - for each $(k, l) \in O(i, j)$ do:
             * output $S_T(k, l)$;
             * if $S_T(k, l) = 1$ then add (k, l) to the end of the LV, code the codevector index, and put the residual vector back into the coefficient array;
             * if $S_T(k, l) = 0$ then add (k, l) to the end of the LV;
           - if $L(i, j) \neq \emptyset$ then move (i, j) to the end of the LIS as an entry of type B;
           - if $(\forall (k, l) \in O(i, j), S_n(k, l) = 0), NoCode(k, l) \leftarrow 1$ else $NoCode(k, l) \leftarrow 0$

      ii. if the entry is of type B then

         - if $(NoCode(k, l) = 0)$ output $S_T(L(i, j))$;
         - if $(NoCode(k, l) = 1$ or $S_T(L(i, j)) = 1)$ then
           - add each $(k, l) \in O(i, j)$ to the end of the LIS as an entry of type A;
           - remove (i, j) from the LIS;

3. **Quantization-step update:**

   - $T \leftarrow 0.65 * T$;

- go to step 2.

The results of ModLVQ show an improvement over SPIHT, for certain types of images. On primarily detailed images compressed with ModLVQ, the PSNR is higher than that of SPIHT by approximately 0.25 dB. For primarily smooth images, however, SPIHT provides better results, often by around the same amount, or more.

2.1.5 vqSPIHT: Variable Quality Image Compression System Based on SPIHT

Another paper that builds on the work of Said and Perlman, and is relevant to this thesis, is [JLN99]. In this work, Jarvi et al. describe a system called vqSPIHT that uses the SPIHT framework for compression, but reserves a certain percentage of the bit budget for coding coefficients that affect the reconstruction of particular areas in the original image.

The target for this algorithm is medical imagery, in particular the images produced from digital mammography. Preserving details in these images is crucial, as even small mistakes in the faithfulness of a decompressed image in relation to the original can result in an incorrect diagnosis. Lossless compression is not an option, due to the large size of these images. A standard size for a digital mammogram is approximately $5,000 \times 5,000$, with 12 bits per pixel (bpp).

An important observation is that some regions of the image are not important for making diagnoses, and thus do not need to be reconstructed as faithfully as others. If a decision can be made as to which coefficients affect the reconstruction of a particular region in the original image, then a portion of the bitstream can be devoted to coding coefficients relevant to the medically important regions of the image.

This decision is made easy due to the nature of the wavelet transform. Each subband of the wavelet pyramid preserves spatial information in addition to frequency information, and thus the location of a coefficient within its subband can be used to determine its corresponding region of effect within the original image.

In order to incorporate this idea into SPIHT, the following modifications are made:

- Before the algorithm is run, the important areas of the image, called "Regions of Interest", must be identified, either automatically, through a feature detector, or manually.

- A threshold α must be determined. This value represents the percentage of the bit budget that will be devoted to normal execution of SPIHT. Once this threshold has been reached, only coefficients that affect the Regions of Interest

are coded, until the bit budget has been exhausted. The value of α is highly application-dependent.

- Normal execution of SPIHT takes place until the percentage α of the bit budget has been reached. At this point, execution of the SPIHT algorithm continues, but coefficients that are not relevant to the Regions of Interest are ignored.

The results achieved by vqSPIHT prove the fundamental idea to be sound. The overall PSNR values of the reconstructed images are similar to those resulting from SPIHT. However, in the Regions of Interest, the PSNR is much higher, by as much as 15 dB for some images, depending on the value of α. Thus, certain regions of the image can be reproduced faithfully, without sacrificing much of the quality of the entire reconstructed image.

2.1.6 Quadtrees

One common data structure that Hy-Q will make use of for image segmentation is the quadtree. Quadtrees have been described as "recursive rectilinear tessellations" in such papers as [WF95]. Despite this complex name, quadtrees are actually quite simple. A quadtree is a data structure formed when an object, such as an image, is divided into four equally sized regions. The process can be continued by dividing some or all of the four new regions into four equal pieces, continuing until no further subdivision is necessary. The result is a tree in which each node can have zero or four descendants, and which segments the image into square regions.

Note that it is possible to build a quadtree in a different way from the one described above. An image, for instance, can be first divided into its individual pixels. Parent nodes in the quadtree can be created by merging four neighbouring pixels, and then, subsequently, four neighbouring nodes. This is called the bottom-up approach, as opposed to the top-down approach.

Some criterion must be used in order to determine whether or not to subdivide a given quadtree square further. One simplified example from the realm of image compression is demonstrated in Figure 2.10.

The criterion used for subdividing is the homogeneity of the current quadtree square. If the pixels in a quadtree square are not all of the same colour, the square is subdivided. Once this process has been completed, the compressed output file is generated by coding the quadtree and the colour of the root nodes.

The advantage of the quadtree is that it is remarkably efficient to code. The drawback is that it can often be too rigid a segmentation strategy to account effectively for all of the homogeneous regions in an image.

The following are some examples of how quadtrees have been used in the image compression literature:

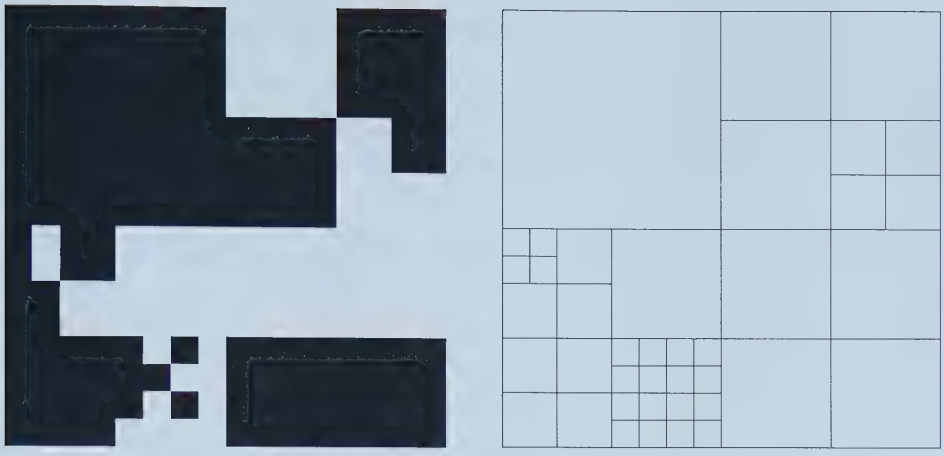


Figure 2.10: Quadtree example: (a) Original Image (b) Quadtree

SR Compression

Another project introduced in [Kni96] builds on the work done by Shusterman and Feder in [SF94]. This method builds quadtrees using the bottom-up approach. However, the quadtree nodes need not all have the same intensity. Instead, nodes are merged if their values do not differ from the average of those nodes by more than a certain threshold. An important consideration made by this algorithm is that it is more significant to merge nodes higher in the quadtree, since these nodes represent more pixels in the original image than do lower nodes. Thus, the initial threshold is halved at each successively higher level of the tree. Once the quadtree has been created, the values of the leaf nodes are quantized to a certain level, according to a formula calculated based on the desired compression ratio.

When the image is being reconstructed, it is important to perform some smoothing, in order to eliminate the blockiness that comes with using the quadtree approach, which assigns a single pixel value to square regions of the image. Shusterman and Feder apply a 3x3 mask in a certain manner that propagates the values of the quadtree leaves down to the pixel level, an idea introduced in [Str90]. Knipe's improvement upon this approach is to use a 2×2 filter instead. The reason for this change is that when a child node at a lower level is being reconstructed, using a 3x3 filter at the parent level will take into account square regions that do not border on the region being constructed. The 2×2 filter limits the regions that are included in the reconstruction to those that do border the region of interest.

Adaptive Wavelet Image Block Coding

The quadtree compression method described above works entirely in the spatial domain. A different method, introduced in [Uhl96], uses information from the original pixels of the image as the criterion for creating the quadtree, but then applies wavelet transforms.

The approach here is top-down. The image is first divided into n blocks, each of which is treated separately. Then each block is divided into four subblocks. The variance of these four subblocks is computed. If the variances are roughly the same, then no further subdivision is done. Instead, each subblock is compressed using wavelet compression methods, but using a different wavelet filter, specially selected from a pre-existing library of wavelet filters in order to minimize certain cost functions based on the properties of the subblock. If the variances of the four subblocks are widely different, then the subblocks are themselves subdivided, with the process described above being applied recursively. If this process arrives at a block size of 2×2, the four pixels are coded directly.

In order to reduce blockiness in the reconstructed image, Uhl applies two strategies. The first is to include extra border information around the subblocks in the wavelet transform, and then average the reconstructed values in the overlapping regions. The other is to use some postprocessing ideas from JPEG, including averaging or smoothing of the border regions.

One important feature of this algorithm is that it is designed with parallel processing in mind. Since the subtrees in the quadtree do not need to interact with each other in their processing, it is easy to use multiple processors, one to work on each subtree. This allows for dramatic improvements in the processing time for the entire algorithm. The results from this algorithm are reasonable, but not up to the level of SPIHT, which was published in the same year.

2.1.7 State of the Art

Since SPIHT was introduced in 1996, many algorithms have been proposed that achieve superior numerical and visual results. The following is a sampling of those algorithms.

Activity Selective SPIHT coding Ramos and Hemami, in [RH99], improve upon some features of SPIHT. They observed that the coefficient ordering introduced by SPIHT puts too much insignificant information at the beginning of the bit stream. Their algorithm assigns weights to the different resolution levels of the coefficient hierarchy, forcing more of the significant information from the coarser resolution levels to be coded first. The result is that the reconstructed images, especially at lower bit rates, are much more recognizable than SPIHT's. This algorithm also uses

results from studies of the Human Visual System to order the coefficients, instead of using the standard approach of trying to minimize the Mean Squared Error. The final feature of this algorithm is that it tries to shift reconstruction artifacts into the detailed portions of the image, since the human eye is much more sensitive to errors in smooth portions of an image.

JPEG-2000 One of the most widely used image compression standards is JPEG, created by the Joint Photographic Experts Group. Based on the Discrete Cosine Transform, JPEG divides an image into small blocks, transforming, quantizing and coding each of them separately. This algorithm suffers, however, from a few problems, in particular the inability to specify an exact bitrate, and some blockiness in the reconstructed images.

JPEG-2000 is being developed in order to bring the JPEG standard up to date. This standard is based upon wavelet transforms, which have proven to be much more powerful than the DCT. It grew out of another wavelet image compression algorithm named CREW ([ZASB95]). The proposed algorithm for JPEG-2000 works as follows: Each image will be subdivided into rectangular, nonoverlapping tiles. Each tile will then be transformed using wavelets and subsequently quantized. The quantized subbands in each tile will then be subjected to a "packet partition", in which each subband is divided into further tiles, or "code-blocks". These code-blocks will then be symbol-encoded.

The JPEG-2000 standard will allow much better performance than the old JPEG, especially at low bitrates. Among its many other features is the ability to allow random access to the bitstream, which will permit editing to be performed on a particular part of the image, without decompressing the entire image.

Many sources of information exist on JPEG-2000. A good high-level summary is given in [MGBB00].

Ridgelets and Curvelets Recently, work has been focused on finding a transform technique more suited to image compression than wavelets. The motivation for this investigation is that wavelets are highly suited to modelling point discontinuities, but have difficulty handling higher dimensional discontinuities, such as edges. Since images are defined by their edges, the goal was to find another type of transform to provide a more compact decomposition of an image than wavelets.

In his doctoral thesis [Can98], as well as in [CD99], Candes introduced the concept of a ridgelet. This is a function that can easily model linear discontinuities such as edges in an image, which serve as a ridge between the higher intensity pixel values on one side and the lower intensity pixels on the other. Later on, curvelets were developed, which can model edges that are curved instead of straight.

In [DV00], Do and Vetterli investigate putting the ridgelet transform into an image compression framework. Their results are quite positive, and indicate that improved compression using ridgelets is feasible.

2.2 Choosing One Algorithm Over the Other

As mentioned in Section 2.1.4, it was found experimentally that ModLVQ generally works better on images that are detailed, and SPIHT works better on images that are smooth. Since these algorithms share the same code for the wavelet transforms and for symbol encoding, it would be useful if a means could be found to identify which quantization strategy would produce better results. Based on this decision, the code for either SPIHT or ModLVQ could be used for the quantization step of the compression process, once the wavelet coefficients have been generated. The following section will introduce a measure that can be applied by Hy-Q to make this decision.

2.2.1 Detail vs. Smoothness

In order to arrive at a suitable measure, it is necessary to understand what properties of an image are responsible for making SPIHT or ModLVQ the better choice as the quantization strategy. The extent to which an image is smooth or detailed seems to play a role, and indeed it does, but the final answer lies with the wavelet coefficients.

As was seen in Section 2.1.1, the distribution of wavelet coefficients is different in detailed images as opposed to smooth ones. In detailed images, the number of significant coefficients (i.e. those that are far from zero) is higher, and furthermore, these coefficients are larger than the significant coefficients in smooth images. These two factors will make a difference to the efficiency with which SPIHT and ModLVQ can code each block of coefficients.

This efficiency can be compared by looking at different types of 4×4 blocks of coefficients. The size 4×4 is selected as the smallest size that allows at least one of each of a SPIHT and ModLVQ node to be examined. Four different types of blocks will be examined here, as follows. Note that the figures quoted with respect to the number of thresholds at which a node is coded are taken from experiments run at a compression rate of 1 bpp. Similar smaller figures apply at lower bpp values.

Case 1: Almost all coefficients are large

The most extreme example of this case comes when all 16 coefficients in a 4×4 block are significant. Experiments have shown that on average, ModLVQ will code a vector matching this block at six different thresholds. Each such step requires 14 bits, 1 to indicate that the vector is significant, and 13 for the index of the vector

from the codebook, which contains 8,192 vectors. Thus the total number of bits required will be 84.

SPIHT will have more difficulty handling a block of this type. 16 bits must be expended to code the signs of the coefficients. Similar experiments to the ones performed for ModLVQ show that on average, six thresholds are required to code each significant coefficient, so this will require another 6 * 16 = 96 bits, for a total of 112.

Case 2: Around half of the coefficients are large

Assume that exactly half of the coefficients are significant, and that there is at least one significant coefficient in each 2×2 subblock of the 4×4 block. ModLVQ will require on average five thresholds to code this block, resulting in 70 total bits.

SPIHT will require 8 bits for the signs of the significant coefficients. Five thresholds are required on average for each coefficient, which contributes 40 more bits. Further bits are required to code the insignificance of the rest of the coefficients, on average 1 per 2×2 block per threshold. The total, then, is 68 bits, which is in the same range as ModLVQ.

Case 3: Only one or two coefficients are large

In this case, SPIHT has a large edge over ModLVQ. The few large coefficients in the 4×4 block will force ModLVQ to code the entire block once the length of the 16-dimensional vector exceeds the current threshold. Because there are only a few significant coefficients, this threshold will likely be low, but nonetheless, each threshold requires 14 bits. Experiments have shown that on average, vectors of this type are coded at 4 different thresholds. This will require 56 bits.

On the other hand, SPIHT can code this type of block with much less effort. Assume that there are two coefficients that are significant, and, to emphasize the worst case, that these coefficients are in different 2×2 blocks, along the lines of Figure 2.11. This leaves two 2×2 blocks that can be coded simply with Isolated Zero or Zerotree Root symbols. On average, as revealed by experiments, blocks of this type will be on the LIS for four thresholds. The two insignificant blocks can thus be dealt with in 8 bits. The significant coefficients will require 5 bits each, one for the sign, and one for each of the four thresholds at which they will be coded (on average). The other insignificant coefficients will require in total an average of 3 bits per threshold. Thus 2*4 + 2*5 + 3*4 = 29 bits are required, significantly below what is required by ModLVQ.

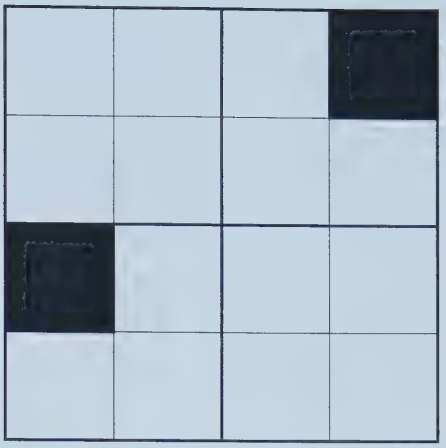


Figure 2.11: Example 4×4 coefficient block - black squares represent large coefficients, white squares represent small coefficients

Case 4: No coefficients are large

This case is the simplest of all. When no coefficients in a 4×4 block are significant, the only symbol that needs to be coded for either algorithm is Isolated Zero (or, depending on the other coefficient blocks in the same subtree, Zerotree Root). In this case, ModLVQ is slightly more efficient, since it will have to code only 1 symbol to 4 for SPIHT.

The figures presented above are simply estimates. Nevertheless, it can be seen from studying the trends in the four cases that as the number of significant coefficients in the hierarchy of an image increases, ModLVQ is able to code the coefficients more efficiently than SPIHT. Since detailed images have more significant coefficients than smooth images do, it makes sense that ModLVQ will produce better results on detailed images, and SPIHT will perform better on smooth images.

2.2.2 Difficulties with Measuring the Coefficients

Unfortunately, choosing between ModLVQ and SPIHT is not as easy as simply calculating how many large coefficients exist in the wavelet hierarchy, and how they are distributed throughout the 4×4 blocks of coefficients. A number of different factors play a role in determining which algorithm will be successful, as follows:

- The location of the significant coefficients is important. If all of the significant coefficients are at the top of the pyramid, then large groups of insignificant coefficients at the bottom of the hierarchy can be coded efficiently with Zerotrees. On the other hand, if significant groups of coefficients exist near the bottom of the hierarchy, then many 4×4 blocks in the higher levels of the tree

will have to be added to the LIS in order to access the significant coefficients, and thus more bits will have to be expended to code the (in)significance of these nodes.

- In ModLVQ, the codevectors chosen to match a given vector of coefficients are crucial. If a codevector exists in the codebook that matches a vector of coefficients nearly exactly, no further vector matchings will be necessary, and thus bits will be saved. On the other hand, of course, it is possible that more bits than usual will have to be expended if the right codevector is missing from the codebook, resulting in a large residual error that must be coded at the next threshold.

- Along the same lines, the initial threshold chosen for ModLVQ makes a big difference to the performance of the quantization. This value, which is calculated as the length of the longest vector in the coefficient hierarchy, determines all of the other thresholds used in the quantization process. A particular initial threshold may result in much better vector matchings than others, and thus require fewer bits to quantize a vector to a satisfactory level of precision. This point was confirmed through experimentation. On each image in my test image set, compression was performed with the initial threshold modified by values selected at regular intervals from the range [-200..200]. In the most extreme case, the PSNR of the reconstructed image was changed by as much as 0.45 dB.

Given this information, it would seem that in order to see how well ModLVQ will quantize a given set of coefficients, it is necessary to determine the initial threshold to be used for ModLVQ, and then try to predict which codevectors will be used. In this way the final residual quantization error from ModLVQ can be obtained, and compared to the same value from SPIHT.

However, this approach is infeasible. The major drawback is that in order to predict with any degree of accuracy, each algorithm must be simulated in its entirety, for the following reasons:

1. It is impossible to guess how many thresholds will be reached, and thus how many codevector matchings will be performed. A similar argument applies to SPIHT. For instance, on the sample set of images used for testing in this thesis, the number of thresholds at 1 bpp ranged from 7 to 20. Making matters even worse is the fact that wide variations in the number of thresholds used was found even among images with similar smoothness characteristics.

2. Even if the number of thresholds can be accurately determined, it must be known, for each vector under consideration, whether or not it will be matched

in the last Sorting Pass. Without running the algorithms, this cannot be known.

Running both algorithms in order to determine which is better would completely defeat the purpose of having a measure to choose between them, as it would take too much time. Fortunately, an alternative is available.

2.2.3 A Proposed Measure of Coarseness

Successful compression involves making tradeoffs. Making an extremely accurate prediction would take far too much time, as discussed in the previous section. It should be possible, however, to sacrifice some accuracy for a measure that would be easier to calculate. The following is a list of some general possibilities that were considered:

- calculate the number of 4×4 coefficient blocks with different numbers of significant coefficients
- calculate the variance of 4×4 coefficient blocks
- calculate the entropy of the coefficients
- use an edge detector to calculate the number of edge pixels in the image

It is worth noting that this last idea is one that does not work directly on the coefficients, but rather on the original image pixels. The principle behind the idea is that large wavelet coefficients arise from sections in the image that have large pixel variation. This is also the basis of an edge detector, which attempts to find the parts of the image in which a transition is made from one intensity of pixel to another. Experiments showed that using the results of an edge detector gave much better predictions as to which quantization algorithm to use than any measure based on the coefficients.

Many different approaches have been developed to detecting edge pixels. One of the most accurate is the Canny edge detector, introduced in [Can86]. The results from using this approach for our problem were not very good, however. The reason is that the sophisticated algorithm behind the Canny edge detector ignores pixel variation that results from noise as opposed to edges that can be identified by the human eye. However, even "noise", which may be a legitimate part of the original image, can result in large wavelet coefficients, and thus it is important to acknowledge this contribution.

One very popular approach to edge detection is to apply a set of one or more masks to each pixel in the image. Common masks, as are often listed in textbooks such as [Cas96], include the Sobel, Laplacian, Prewitt and others. All of these gave

reasonable results, but the best results came from the Kirsch masks, which are given in Tables 2.8 and 2.9.

-3	-3	-3
-3	0	-3
5	5	5

Table 2.8: The horizontal Kirsch edge detection mask

-3	-3	5
-3	0	5
-3	-3	5

Table 2.9: The vertical Kirsch edge detection mask

In order to calculate a value for a particular pixel, each mask is applied with its centre at that pixel. The two resulting mask values are the horizontal and vertical components of the gradient of the image change at that pixel. The response from the edge detector is the magnitude of the gradient. The centre pixel is considered to be an edge pixel if the gradient magnitude is above a certain threshold.

The measure chosen for quantization strategy selection involved using this Kirsch edge detection value to determine how many pixels in an image are "edge" pixels. For this, a threshold was needed. Experiments showed that a value of 18 gave the best results. This is quite a low number, causing the edge detector to be sensitive even to small amounts of noise. As explained above, however, even small pixel variation can lead to large wavelet coefficients. This new measure will be named Kirsch18.

The final decision that needs to be made is how to choose between ModLVQ and SPIHT based on Kirsch18. Complicating this choice is the fact that for a given image, one strategy is not always best. Consider Table 2.10, which shows that for the famous test image Lenna, neither quantization strategy is better at all bpp values.

Bits Per Pixel	Better algorithm
1.0	ModLVQ
0.5	SPIHT
0.25	SPIHT
0.125	SPIHT
0.0625	SPIHT

Table 2.10: The better algorithm at various bpp values for test image Lenna

It would seem that a different decision would need to be taken at each bpp value. This would be extremely difficult, especially since experimental data revealed no link between properties of the edge detector, and the bpp values at which one algorithm becomes more effective than the other. A compromise is necessary, and it is to follow the decision arrived at from examining the results of one particular bpp value. After many experiments, the most suitable bpp value was found to be 1.0. Figure 2.12 shows the graph of plotting the results of the difference in PSNR from compressing each image with ModLVQ as opposed to with SPIHT against Kirsch18. It was found that for any image with a Kirsch18 value smaller than 0.7, SPIHT was **always** the better quantization strategy, and thus the smoothest images from the test data set were not used for this comparison, to avoid skewing the results.

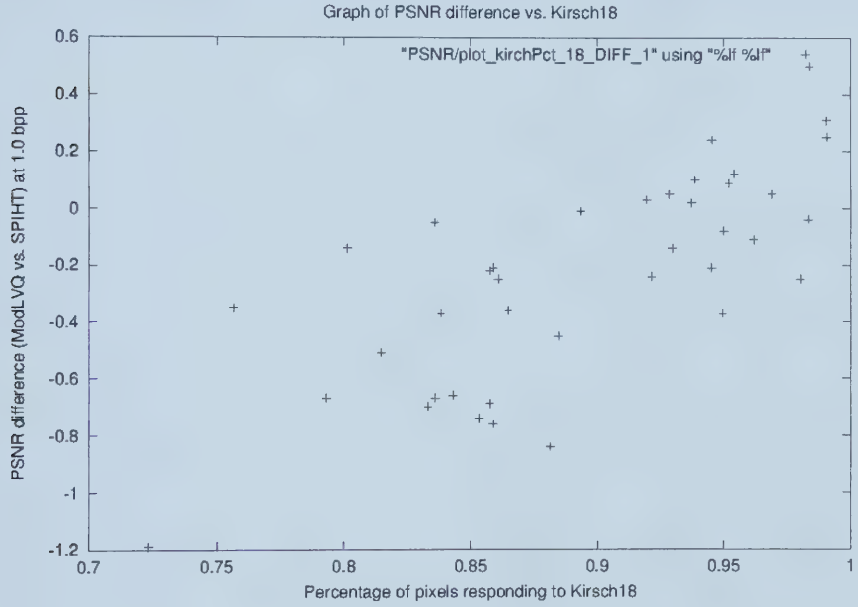


Figure 2.12: Graph of PSNR difference vs. Kirsch18

The correlation between these two variables is 0.76. Statistically, this represents a reasonably strong, but far from perfect correlation, which confirms the visual impression conveyed by the graph. Using further statistics such as computing the line of best fit for these data, and some adjusting based on experiments, the threshold for choosing between SPIHT and ModLVQ was set at 0.94. In other words, images in which 94% of the pixels are identified by Kirsch18 as being edge pixels should be quantized using ModLVQ, and images with fewer than 94% such pixels should be quantized with SPIHT.

The effectiveness of Kirsch18 was tested on a set of 43 test images, at five different bpp values. The results are given in Table 2.11.

Number of correct decisions (out of 5)	Number of images (out of 43)
5	27
4	4
3	5
2	3
1	1
0	0

Table 2.11: Determining the effectiveness of Kirsch18

Out of 215 decisions, Kirsch18 made the correct one, i.e. chose the quantization strategy that led to the higher PSNR, in 184 of them, or 85.58%. Given the difficulties identified above in trying to arrive at a perfect measure, this is quite a reasonable result.

2.3 Combining Quantization Strategies

The Kirsch18 measure discussed in Section 2.2 allows Hy-Q to choose one quantization strategy for compressing the entire image. The majority of the time, this is the strategy that leads to a higher PSNR when reconstructing the image. The result is an algorithm that is effective on both smooth and detailed images, and is thus more useful than either SPIHT or ModLVQ alone.

In practice, however, most images are not uniformly smooth or uniformly detailed. They contain some regions of large pixel variation, and some regions that have nearly constant pixel values. This leads to the natural question of whether or not both quantization strategies can be used in the same image. Could ModLVQ's vector quantization be used for the detailed portions of the image, and SPIHT's scalar quantization be used for the smooth portions? The answer is yes, and the rest of this section is devoted to discussing the various issues associated with implementing this idea.

2.3.1 Subdividing the Image

The first decision to be taken by the algorithm involves how to break up the image into its smooth and detailed regions. The strategy chosen by Hy-Q is to subdivide the image along quadtree boundaries. A discussion of the advantages and disadvantages of quadtrees in general can be found in Section 2.1.6.

To illustrate the efficiency of the quadtree representation, consider a 512×512 image compressed to 1 bit per pixel. The bit budget for the compressed file is 512 * 512 * 1 = 262,144 bits. If only one level of quadtree subdivision is performed, the

following bits need to be coded into the output file:

- 1 bit to indicate that the original image should be subdivided
- 4 bits, one to represent the quantization strategy used for each of the four quadrants (there are only two possibilities, SPIHT and ModLVQ, so one bit is sufficient to represent the choice in each quadrant)
- 4 bits, one for each quadrant to indicate that it will not be subdivided further

The total for the entire quadtree is 9 bits, or 0.0034% of the total bit budget.

Other, more sophisticated segmentation methods could also be used to subdivide the image. For instance, Wu and Fang, in [WF95], present a compression scheme that uses Binary Adaptive Segmentation (BAS) trees to divide an image into a set of "canonical" polygons that have at least 3 but no more than 8 sides. Each polygon contains an image segment of nearly constant pixel values. This process could be used in Hy-Q to locate the smooth regions in the image, designating any regions not accounted for by a polygon as detailed.

In the end, however, it was decided to forgo using these more advanced techniques in favour of quadtrees, for the following reasons:

- The cost, in terms of bits used in the output file, of coding the more sophisticated subdivision could be prohibitive. For instance, with a 256×256 image compressed to a ratio of 128:1 (0.0625 bpp), the entire output file is only 256 * 256 * 0.0625 = 4,096 bits. The hundreds, perhaps thousands of bits needed to represent a complex subdivision would starve the other elements of the coding process, and lead to poorer compression results.
- The time needed to arrive at the subdivision would likely be too long. In order for a compression scheme to be accepted, it must run in a reasonable time. While complex segmentation schemes have been demonstrated to be fairly efficient, they cannot come close to the simplicity of the quadtree.
- The quadtree representation fits nicely with the descendant structure of the coefficients in the wavelet hierarchy, as each coefficient has four descendants, just like a quadtree node. A different segmentation strategy might cause some problems in this respect. This will be made more clear in Section 2.3.3.

2.3.2 Coding Image Sections as Separate Images

Now that a quadtree is being used to divide the image into quadrants, the next decision required by Hy-Q involves how to apply the right quantization strategy, and only that strategy, to the right quadrant. Perhaps the most obvious way to do this

would be to treat the quadrants as separate images, compressing them individually. Consider the image House, shown in Figure 2.13.

Figure 2.13: Test Image: House

The results of measuring each quadrant, as shown in Table 2.12, lead to the four sub-image/quantization strategy pairs displayed in Figure 2.14.

0.72235	0.73406
0.91352	0.96876

Table 2.12: Kirsch18 values for the four quadrants of House

Figure 2.14: The four quadrants of House, with their quantization strategies

Each subimage is allocated one quarter of the entire bit budget (minus the various administrative bits that need to be coded, such as the size of the image). As each subimage is compressed separately, using its specified quantization strategy, its compressed output is appended to the overall output file for the entire image. This algorithm produces correct output, but was rejected for the following reasons:

- The most obvious criterion by which to judge a compression algorithm's performance is the quality of its reconstructed images. The results from this

algorithm were below those of SPIHT and ModLVQ, by up to 0.2 dB PSNR. Several small factors influence this result. The first has to do with the number of coarse coefficients in the wavelet hierarchies. Since there are now four images to compress instead of one, the number of coarse coefficients that form the root of a hierarchy increases four-fold. These coarse coefficients are much larger than the detail coefficients, and thus take more bits to compress, which affects the number of bits that are available to code the rest of the coefficients. The other factor, and likely a more important one, concerns the increase in the size of the boundary. When the wavelet transform is applied at the boundary of an image, part of the wavelet filter will "hang off the edge" of the image. To remedy this situation, the image is artificially extended using one of several possible strategies, such as reflecting coefficients about the edge of the boundary (a detailed discussion of reflection can be found in Section 3.3). This extension is necessary, but the result is that the coefficients along the boundary of the coefficient hierarchy do not accurately represent the local properties of the image at its boundaries. When four sub-images of the original image are used, the number of boundary pixels increases by a factor of roughly two, and these negative effects become more prominent. Figure 2.15 gives a good indication of how many extra boundary coefficients exist.

- Using this approach means that the ability to do progressive transmission is lost. Progressive transmission refers to the process by which information is sent in decreasing order of importance. For instance, if an image is downloaded from the Internet, especially with a slow connection, a coarse version of the original image is the first thing that is seen, followed by more and more detailed versions until the entire image has been downloaded. It is important for the user to be able to tell generally what the image looks like, so that they can determine whether or not to stop the download, or wait for the detail. It wouldn't make sense to send the small detail information first, as it would be incomprehensible without the context of how the image looks in general. Algorithms such as SPIHT have the progressive transmission property. Since the coefficients are coded in nearly sorted order from the largest to the smallest, it can be guaranteed that if a transmission of a SPIHT-compressed image is stopped part-way through, the image that can be reconstructed using only the transmitted bits is the best possible. The result of using the above approach would be that first the coarse version of one quadrant would appear, followed by all of the detailed information for just that quadrant. The user would have to wait for almost the entire compressed data stream before they could tell what each quadrant of the image looks like.

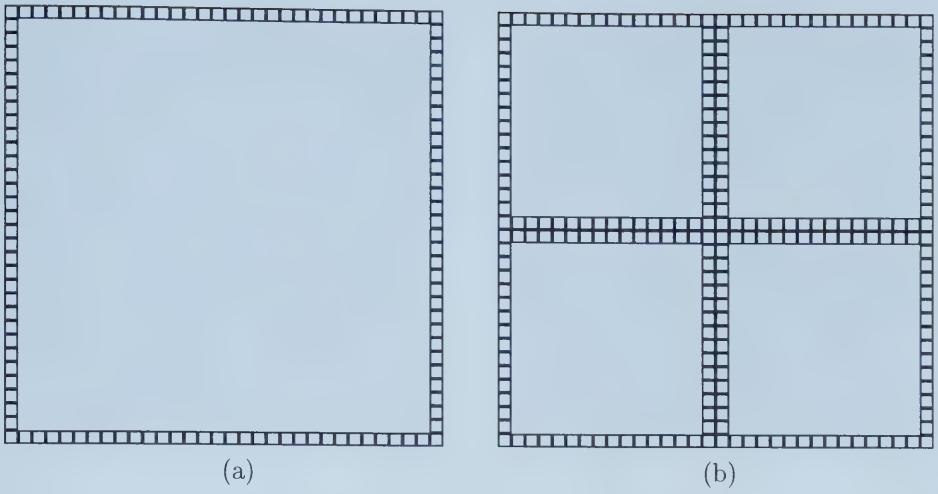


Figure 2.15: Boundary pixels: (a) from SPIHT (b) from this section's algorithm

2.3.3 Compressing Image As A Whole, But Using Two Passes

In order to solve the problems created by the first version of the algorithm, it is necessary to perform the wavelet transforms on the entire image, instead of four separate transforms, one on each quadrant. This means that the scalar and vector quantization would have to be performed on the same hierarchy of wavelet coefficients. Since SPIHT divides the coefficients into blocks of size 2×2, and ModLVQ makes blocks of size 4×4, it would seem necessary to make two separate passes, in one pass using SPIHT to code the coefficients from the smooth areas of the image, and in the other pass using ModLVQ to code the coefficients from the detailed areas of the image. The question then arises as to whether or not it is possible to determine from what region of the original image a particular set of coefficients is related.

The answer to this question is yes, due to a property of the wavelet transform. In addition to providing frequency information, wavelet coefficients also preserve spatial information. Consider Figure 2.16, which shows the coefficient hierarchy after the famous test image Lenna has been transformed. Note that each pixel value represents the magnitude of the coefficient at that location, and that the smaller pixel values have been magnified in order that they may be visible.

The figure shows that each subband contains a picture that looks similar to the original Lenna. The resolution 0 subband, in the top-left corner, contains the coarse coefficients, and thus has similar intensities to the original image. The other subbands, which contain the detail information, are black in smooth regions and non-black around the edges in the original image, creating a ghostly image that outlines the features of Lenna.

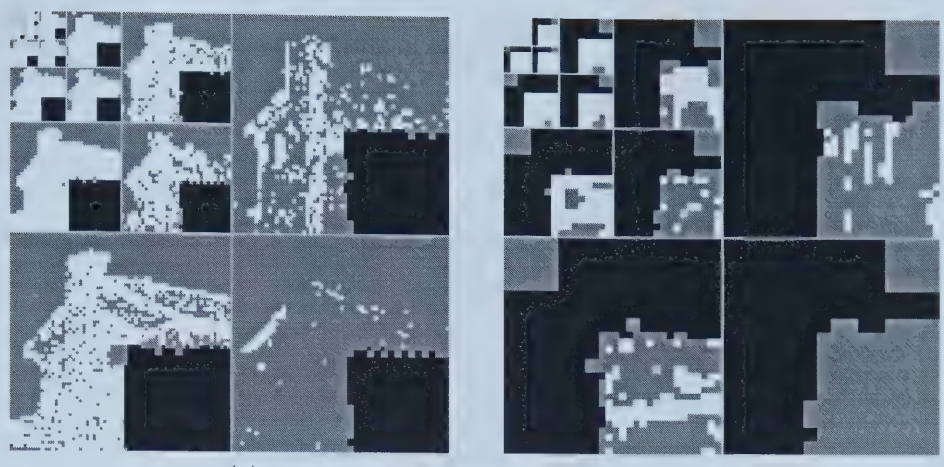
(a) (b)

Figure 2.16: Lenna: (a) original image (b) after wavelet transforms

With this information in mind, it is easy to determine, for a given coefficient, its relationship to the original image. Its location and size are found by scaling the subband to the dimensions of the original image. This approach was motivated by the discussion of Regions of Interest in [JLN99]. This paper is discussed in more detail in Section 2.1.5.

In order to visualize this process, consider Figure 2.17. Its two images show maps of the wavelet coefficient hierarchy for the image House, one for the SPIHT coding pass, and one for the ModLVQ pass. White pixels represent coefficients that are coded, black pixels represent coefficients that are rejected as being unrelated to the current region of interest, and grey pixels represent coefficients that were never considered by the algorithm. Notice that for the SPIHT image, the bottom right corner of each subband is rejected, and that in the rest of each subbands, the coefficients that are coded come from the areas containing the house. In the ModLVQ image, three quarters of each subband is rejected, while the bottom right corner is densely coded (except in the highest resolution subbands), since there is a great deal of detail in that corner of the original image.

The modifications to SPIHT required to implement this algorithm are simple. The original algorithm is run twice, once for coding with SPIHT, one for coding with ModLVQ. In the Sorting Pass of each run, descendants of each node on the LIP are tested for significance at the given threshold. An additional test is added to see if the descendant is relevant to the portion of the original image being coded with the current quantization strategy. If the descendant is not related, then it is not added to the LIS, even if it is significant. This eliminates not only that direct descendant, but also all of its descendants, which are relevant to the same portion



(a) (b)

Figure 2.17: Coefficient hierarchies: (a) SPIHT pass (b) ModLVQ pass

of the image. The compressed output of the second run is appended to that of the first run to create the final output file. When the image is reconstructed, the reverse wavelet transforms are applied first to the coefficients coded by SPIHT, and then to the coefficients coded by ModLVQ. The subsections that were originally coded by each algorithm are extracted from their respective reconstructed image, and glued together to form the final output.

The results of running this version of the algorithm improved upon those of the previous version, but still didn't reach the results from SPIHT and ModLVQ separately. Again, there is a boundary problem. This time, however, it is not the boundary around the image that is the problem, but rather the boundary between the different regions marked by the quadtree.

Going back to the example image House, we saw in the previous section that the results of Kirsch18 showed that the three quadrants in the top and left of the image should be coded with SPIHT, and the bottom right quadrant with ModLVQ. For the SPIHT pass, only the coefficients outside the bottom right corner of each subband are coded. This leads to a significant problem, however. The length of the wavelet filter (up to 9 numbers for D97, the filter used in SPIHT) means that when a coefficient at the boundary of a lower subband is being reconstructed, coefficients in the higher subband that are in the ModLVQ region must be available. Since these are not coded in the SPIHT pass, and thus not available for reconstruction, boundary artifacts arise. For instance, Figure 2.18 shows, in close-up, the line artifact that clearly demarcates the quadtree boundary between the SPIHT and ModLVQ regions, as well as the "ringing" around the edges defined in the image.

The only solution to this problem presents its own difficulties. It is to code a

Figure 2.18: Boundary artifact

few extra coefficients around the boundaries of the regions of interest in each pass. In this way, all of the coefficients needed to perform the inverse wavelet transform are available in the output stream, and thus the reconstructed boundaries are free of artifacts. The problem lies in the fact that these extra coefficients take up extra space. Crucial bits in the overall bit budget have to be sacrificed in order to solve the problem, and this in turn limits the number of bits available for the rest of the image. The result is that the PSNR of the reconstructed image falls short of the results achieved by SPIHT and ModLVQ alone.

It should also be pointed out that this approach does not solve the problem of progressive transmission, as discussed in the previous section. There are still multiple passes being performed, and thus a coarse version of the entire image cannot be reconstructed until a large portion of the compressed file has been transmitted.

Given the shortcomings still present in the algorithm as presented in this section, it would seem that the ideal solution would be one in which both SPIHT and ModLVQ are used on the same set of coefficients, **during the same pass**. This is indeed possible, and the idea is the foundation of the algorithm that is presented in the next section.

2.3.4 Compressing Image In One Pass - Hy-Q

Managing Descendants

As pointed out in the last section, the key issue preventing SPIHT and ModLVQ from being used at the same time is the size of coefficient blocks considered by each algorithm. Due to the fact that the ModLVQ coefficient blocks contain four times the number of coefficients that the SPIHT blocks do, it would seem that mixing them would lead to confusion and overlap, and thus an inefficient quantization

procedure. However, when considered in conjunction with the descendant structure of the coefficients, this problem can easily be addressed.

Consider Figure 2.19, which shows the descendant structure in both SPIHT and ModLVQ. For both algorithms, each coefficient block has four descendants. The only exceptions are the root nodes, which are a special case and will be described later. An important point is that the four descendants are positioned together, forming a square. In particular, the four 2×2 descendants of a SPIHT node form a 4×4 block, which is exactly the size of a ModLVQ block. Thus, when a SPIHT node should have ModLVQ descendants, the four immediate descendants are merged into one. This can be seen in Figure 2.20.

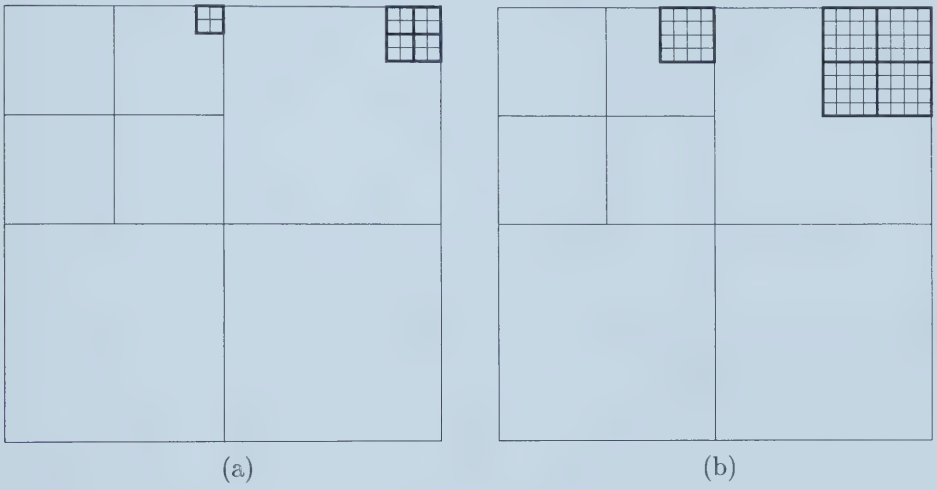


Figure 2.19: Coefficient descendant structure: (a) SPIHT (2×2 blocks) (b) ModLVQ (4×4 blocks)

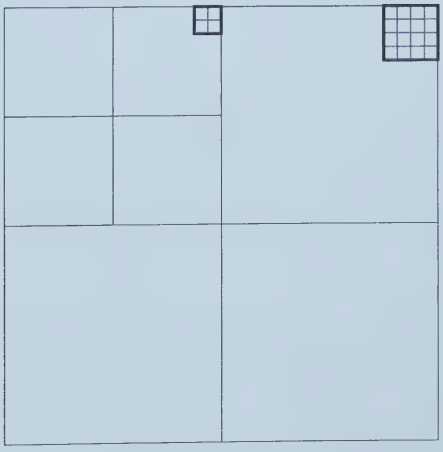


Figure 2.20: 4×4 ModLVQ descendant from 2×2 SPIHT parent

This process is applied in reverse when a ModLVQ node has SPIHT descendants. Each 4×4 ModLVQ block can be subdivided into four 2×2 SPIHT blocks. Thus, each descendant of a ModLVQ node can be replaced by four SPIHT nodes, as seen in Figure 2.21.

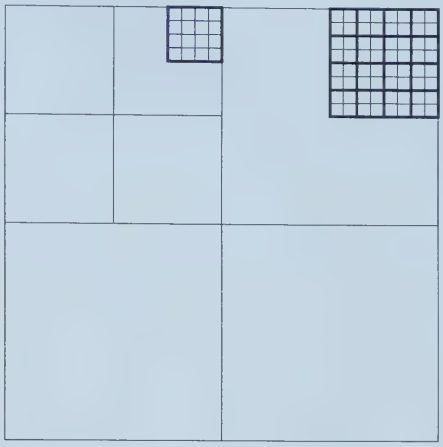


Figure 2.21: 2×2 SPIHT descendants from 4×4 ModLVQ parent

The fact that there are four descendants of each node works perfectly with the quadtree subdivision scheme that is used to divide the image into smooth and detailed regions. Another crucial point is that because of the spatial information contained in the subbands, as discussed in Section 2.3.3, each subband can be divided using the quadtree to determine which quantization strategy should be used to code each block of coefficients. This information can be used to determine what the type of each descendant should be.

For instance, consider Figure 2.22, which shows the descendants of a node in the image House. The parent node is a 4×4 ModLVQ block. The four descendants of this node completely cover the next subband. The quadtree for this image, however, dictates that the top half of the image, and thus the top half of each subband, should be coded with SPIHT, while the bottom half of the image should be coded with ModLVQ. Thus the bottom two descendants are 4×4 ModLVQ blocks, while the top two descendants are split up into four 2×2 blocks, to be coded with SPIHT.

This switch from ModLVQ to SPIHT, or vice versa, is necessary because at the very top level of the tree, the size of the subbands is too small to subdivide along quadtree lines. In the implementation of Hy-Q, the maximum number of levels of wavelet transform is applied. The result is that the two highest resolution subbands of the hierarchy are of size 4×4 (the wavelet filters are of length 9, and thus cannot be applied further). One algorithm must be chosen to code the root nodes, and thus, if the image is to be subdivided, a switch between quantization strategies can be made once the subbands become large enough.

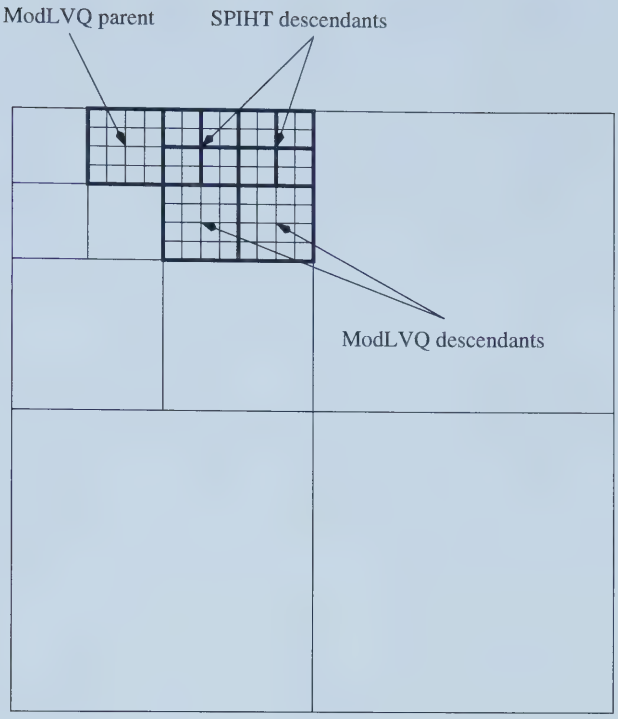


Figure 2.22: Example of a Node With Different Types of Descendants

This issue can manifest itself at lower levels of the hierarchy as well, depending on how many levels of quadtree subdivision are used. For instance, consider a subband that measures 8x8. If only one level of quadtree decomposition is performed, then the subband can be divided into four quadrants of size 4×4, and either ModLVQ or SPIHT can be chosen, as appropriate, to code each quadrant. However, if two or more levels of quadtree subdivision are performed, the subband would be divided into regions of size 2×2 or smaller. In this case, each node used to code this subband would have descendants of both the SPIHT and ModLVQ variety, and thus a switch would have to be made at a lower level of the hierarchy.

The descendants of the root node present a small special case. Whereas most nodes have four descendants, root nodes, as discussed in Section 2.1.3 have only three. These descendants reference the exact same spatial location in the original image as the root node, and thus are coded with the same algorithm as is used to code the root node.

A full picture of the descendant structure can be seen in Figure 2.23. It is based on the image House, although magnified so that the different sizes of coefficient blocks can be seen clearly. Notice how in each resolution level below the top two, the bottom right quadrant is coded with the larger ModLVQ blocks, while the other quadrants are coded with the smaller SPIHT blocks.

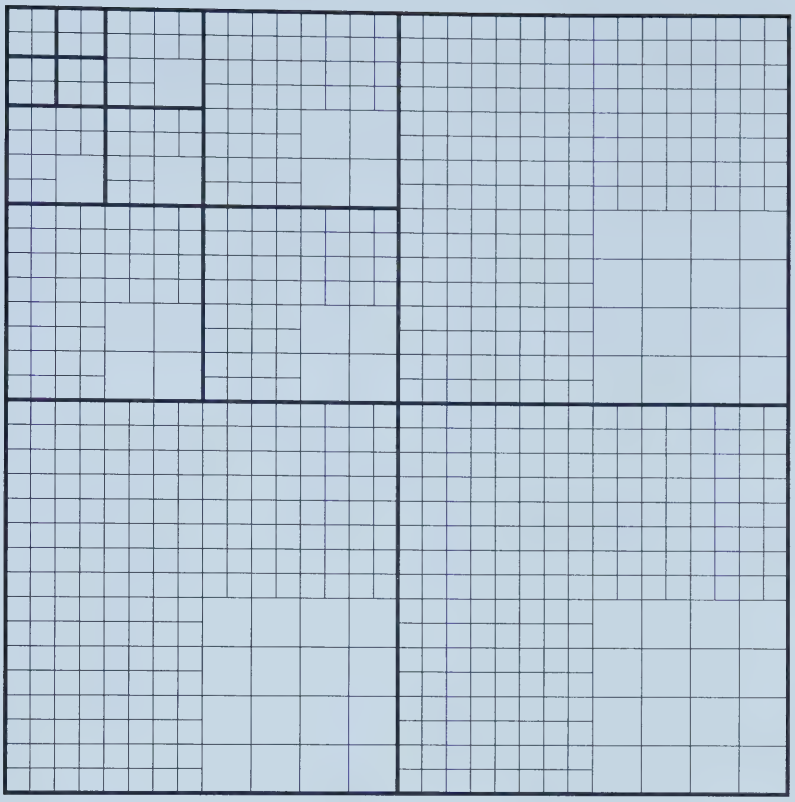


Figure 2.23: Map of quantization strategies in House

The major advantage of this approach is that it eliminates the boundary problems that plagued the earlier versions of the algorithm. The wavelet transforms are applied to the entire image at once, so that the amount of reflection required is as small as possible. Even more important, however, is the fact that no redundant coefficients need to be coded. The main issue for the previous version of the algorithm was making sure that when it came time to perform the inverse wavelet transforms, the coefficients along the boundaries of the regions of interest were available. That problem completely disappears in Hy-Q. The coefficients just outside the boundaries of the SPIHT regions of interest fall in the ModLVQ regions of interest and, of course, vice versa. Thus all of the significant coefficients required for the inverse wavelet transforms will be available, provided that there is room for them in the output file.

Predicting Descendants

The most important aspect of the quantization algorithm, as originally developed by Shapiro in [Sha93], is the identification of significant descendants from a given node. At each threshold, it must be determined which subtrees contain coefficients that are

significant at that threshold, and which subtrees contain no significant coefficients, and thus can be coded simply with a zerotree. In order that this information may be easily accessible, Hy-Q expands on an idea originally implemented in SPIHT.

After the wavelet transforms have been performed and the coefficient hierarchy is in place, SPIHT does calculations to determine, for each node, the magnitude of the coefficient with the largest absolute value in each of its descendant subtrees. When, in the Sorting Pass, the decision needs to be made as to which subtrees of a given node contain significant coefficients, these values, which are stored at the same array indices as the node's coefficients, can simply be compared to the current threshold. Values greater than the threshold indicate that the descendants in that subtree should be added to the LIS, and values smaller than the threshold indicate that the entire subtree can be coded with a zerotree.

This information is stored in a two-dimensional array called max_image. The name comes from the fact that it shows the maximum descendants throughout the entire image. Table 2.13 shows the top two resolution levels of the max_image when SPIHT codes the image House.

3008.29	1556.68	3028.41	776.04	137.27	434.67	159.94	7.80
886.89	777.85	1544.59	815.77	615.97	618.90	776.04	232.22
2659.68	951.82	2310.84	1170.37	951.82	481.19	1170.37	680.92
2030.53	577.96	2416.11	1287.02	640.49	101.57	681.97	233.44
19.57	701.12	844.81	372.75	66.35	683.24	158.41	22.50
365.93	494.85	981.92	1410.66	422.27	529.44	815.77	294.79
328.27	227.51	537.77	429.69	288.44	170.76	263.07	87.80
695.97	198.54	531.96	584.08	223.04	60.80	169.35	483.33

Table 2.13: SPIHT max_image for the image House

Every entry is filled with a value, since each node contains four coefficients, and has four descendants. The root nodes have only three descendants, but for these cases, the fourth value contains the magnitude of the largest coefficient in that root node.

A similar process is applied in ModLVQ. One difference is that because ModLVQ's nodes contain 16 coefficients but still have only four descendants, many of the entries in the max_image are blank. A more important difference, however, is that since ModLVQ codes 4×4 groups of coefficients as 16-dimensional vectors, the entries in the max_image are not the magnitudes of individual coefficients, but rather the lengths of vectors. Table 2.14 shows the top two resolution levels of the ModLVQ max_image for House. The root node is treated slightly differently from the rest, as the maximum value of its descendants are placed in different locations within its 4×4 block. ModLVQ does not record the magnitude of the root vector, and thus the top left coefficient is blank.

0.00	2403.54	0.00	0.00	1051.43	0.00	806.24	0.00
4117.70	1906.73	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	1493.01	0.00	2021.47	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
935.95	0.00	2396.11	0.00	794.24	0.00	1129.75	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
1302.97	0.00	1319.59	0.00	389.04	0.00	579.22	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 2.14: ModLVQ max_image for the image House

It would be tempting to try to keep just one max_image variable for Hy-Q, in exactly the same style as SPIHT and ModLVQ do. However, because a given node can have two different types of descendants, and the information required from each type is different (magnitudes of individual coefficients vs. lengths of vectors), this is not possible. Hy-Q's solution is to keep two separate max_image variables. The entries in the SPIHT_max_image keep track of the largest individual coefficient in each descendant subtree **that will be coded with SPIHT**. Similarly, the entries in the ModLVQ_max_image keep track of the longest 4×4 vectors **that will be coded with ModLVQ**. Tables 2.15 and 2.16 show the two max_image variables for the top two resolution levels of the image House, when SPIHT is used to code the root nodes.

3008.29	1556.67	3028.40	776.04	137.27	434.67	159.94	7.80
886.89	777.85	1544.59	815.77	615.97	618.90	776.04	232.22
2659.69	311.32	2310.84	922.41	951.82	481.19	0.00	0.00
2030.54	577.96	2416.11	1287.02	640.49	101.57	0.00	0.00
19.57	701.12	844.81	372.75	66.35	683.24	158.41	22.50
365.93	494.85	981.92	1410.66	422.27	529.44	815.77	294.79
328.27	227.51	0.00	0.00	288.44	170.76	0.00	0.00
695.97	198.54	0.00	0.00	223.04	60.80	0.00	0.00

Table 2.15: Hy-Q's SPIHT_max_image for the image House

Note that the bottom right corner of each of the lower resolution subbands in the SPIHT_max_image example is blank. This is because, according to the quadtree subdivision, the bottom right quadrant of the image is to be coded with ModLVQ. Thus there are no descendants of these nodes that will be coded with SPIHT. Similarly in ModLVQ_max_image, all but the bottom right corner of each subband is blank. One other interesting point to note is the entry in bold in ModLVQ_max_image. This entry is for a node that will be coded with SPIHT, so it would seem that there should be four entries in the local 2×2 block, for its four descendants. However, the descendant of this node will be coded with ModLVQ, and thus there will be only

0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	2021.47	0.00	0.00	2021.47	0.00
0.00	0.00	1319.59	579.22	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	**1319.59**	0.00	0.00	0.00	579.22	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 2.16: Hy-Q's ModLVQ_max_image for the image House

one descendant, giving just the one non-zero entry.

One result of the different ways in which SPIHT and ModLVQ code the coefficients is that two different thresholds must be maintained. Not only are the vector lengths larger than the magnitudes of the individual coefficients, but the percentage decrease in the threshold after each Refinement Pass is different. SPIHT divides the threshold in half, while ModLVQ multiplies it by 0.65. Thus, Hy-Q keeps two separate thresholds, one for SPIHT and one for ModLVQ.

Dividing the Bit Budget

Because two different quantization strategies are being used, it is necessary to decide how many bits to allocate to each. The most obvious choice would be to divide the bit budget according to the percentage of the quadtree allocated to each strategy. For instance, if one level of quadtree decomposition is used, with two quadrants being allocated to SPIHT and two to ModLVQ, it would seem that 50% of the total bits should be used for coding SPIHT coefficients, and 50% for coding ModLVQ coefficients. This strategy, however, is not necessarily optimal.

To see why this is the case, consider the types of coefficients being quantized by each algorithm. SPIHT quantizes the smooth regions of the image. As was demonstrated in Section 2.1.1, these regions should contain smaller coefficients than the coefficients in the detailed regions, which are quantized with ModLVQ. Evenly allocating the bits would allow the coefficients quantized by SPIHT to be reconstructed with a small amount of error, but would leave a large amount of error in the ModLVQ coefficients. By giving more bits to ModLVQ from SPIHT, the overall error in the reconstructed image can be reduced by sacrificing a small amount of precision in the SPIHT coefficients in order to effect a large increase in precision in the ModLVQ coefficients.

The key decision involves how many bits to reallocate from SPIHT to ModLVQ. In practice, it is difficult to determine the exact value that gives the best result for the whole image. The decreased error resulting from adding more bits to ModLVQ

may be more than offset by a larger increase in error resulting from taking bits away from SPIHT. Fortunately, two facts allow a decision to be taken by Hy-Q. The first is that the curve generated by pairs of bit reallocation values (the number of bits given from SPIHT to ModLVQ) and PSNR values is roughly a parabola, for every image. There are small exceptions, where reallocating more bits results in an unexpected increase or decrease, but the curve does not make any wild deviations. To see an example of this, consider Figure 2.24, which shows a graph of bit reallocation values against PSNR results for the image Shepherd, seen later in Figure 2.27.

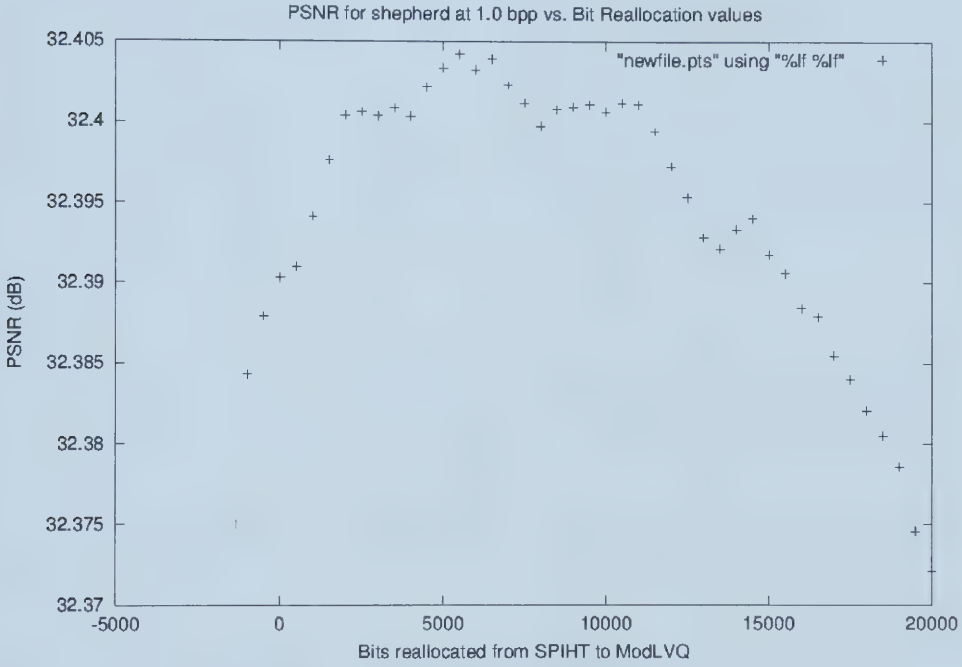


Figure 2.24: Plot of PSNR for Shepherd at 1 bpp vs. Bit Reallocation Values

The other helpful fact is that experimentation has shown that for most images, a large range of values gives nearly equivalent PSNR results for the reconstructed image. For instance, with the image Shepherd, the optimal bit reallocation value at 1 bit per pixel is 5,500, giving a PSNR of 32.41 dB. However, any value in the range of 1,500-12,500 gives the same result up to the second decimal place.

The actual number of bits that should be reallocated is estimated by Hy-Q using information from the quadtree, searching through different possible values, and examining the number of bits used by SPIHT after each Refinement Pass. One feature of Hy-Q is that it allows the user to set the bit reallocation value manually. If one part of the image needs to be reproduced in greater clarity than the others, then more bits can be given to the algorithm that will be used to code that region.

Algorithm Summary

In order to put all of the different aspects of Hy-Q in their proper places, this section will give a summary of the Hy-Q algorithm, in the same style as was given for SPIHT in Section 2.1.3 and for ModLVQ in Section 2.1.4. Rather than repeat those algorithm descriptions, only the aspects of the algorithm that are new in Hy-Q will be given.

1. **Initialization:**

   (a) Apply Kirsch18 to the image to determine whether or not the image should be subdivided, and, if so, how. Output a representation of the quadtree.

   (b) Calculate the magnitude of the largest individual coefficient *to be coded with SPIHT*, and the length of the longest vector *to be coded with ModLVQ*. Output the numbers n for SPIHT and T for ModLVQ based on these calculations.

2. **Sorting Pass:**

   (a) When looking for the largest descendant from a node, examine both the SPIHT_max_image and the ModLVQ_max_image. Test for significance by comparing to the appropriate threshold.

   (b) When making the transition from a parent node quantized with SPIHT to a descendant subtree quantized with ModLVQ, add only one ModLVQ node to the LIS.

   (c) When making the transition from a parent node quantized with ModLVQ to a descendant subtree quantized with SPIHT, add four SPIHT nodes to the LIS.

3. **Refinement pass:** same as in SPIHT (there is no Refinement Pass in ModLVQ).

4. **Quantization-step update:** decrement the n value for SPIHT, and multiply T for ModLVQ by 0.65.

It should be noted that Hy-Q, as it is currently implemented, is fairly time-efficient. When only one quantization strategy is used, Hy-Q is marginally slower than the corresponding algorithm, due to the overhead of calculating the Kirsch18 measure. When both strategies are used, Hy-Q is slightly slower than SPIHT, but faster than ModLVQ. On a Pentium-II 400 MHz computer, compression and decompression of a 512×512 image using Hy-Q takes just over two seconds.

Progressive Transmission

The final point to note about Hy-Q is that it preserves the property of progressive transmission. SPIHT and ModLVQ are mixed into the same bitstream, but instead of having all of the coefficients coded by the one algorithm appear before all of the coefficients coded by the other algorithm, the most significant coefficients from each algorithm appear at the beginning of the bit stream, and the least significant coefficients from each algorithm appear at the end. This allows a coarse version of the compressed image to be reconstructed after only a small number of bits have been decoded from the bit stream.

It should be noted that the bit stream is not guaranteed to be perfectly divided between SPIHT and ModLVQ. It might be expected that among the first N bits of the compressed file, the number of bits from each algorithm should correspond to the percentage of the quadtree allocated to that algorithm, thus allowing all sections of the image to be reconstructed to approximately the same level of detail. This, however, may not be the case.

The reason lies in the initial threshold used by each algorithm. For instance, consider an image in which the root coefficients are much larger than any other coefficients in the image. If ModLVQ is being used to code that group of root coefficients, the initial threshold for ModLVQ will be so large that it will take several iterations of the algorithm, each with a Sorting and a Refinement Pass, for the ModLVQ threshold to decrease to the point at which other ModLVQ nodes can be coded. In the meantime, many SPIHT nodes may have begun to be coded, with the result that SPIHT takes up much more of the beginning of the bit stream, and not as much at the end.

Thus the transmission is nearly, but not completely, progressive. It is important to notice, however, that even SPIHT is not perfectly progressive. Because SPIHT is based on a heuristic, the coefficients are not coded in perfectly decreasing order of significance. It is true that all coefficients whose magnitude is greater than a given threshold are coded before the coefficients whose magnitude is smaller than that threshold, but even in this group, there is no guarantee of sorting.

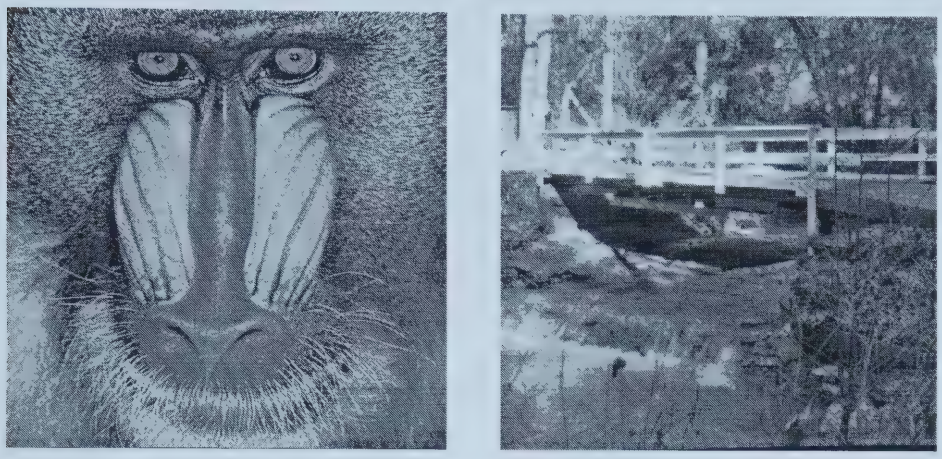
2.4 Results

The results obtained from running Hy-Q on the images in my sample set are encouraging. They can be divided up into three categories, as follows.

2.4.1 Uniformly coarse or smooth images

Some images cannot be divided into detailed and smooth sections along quadtree boundaries. In this case, Hy-Q chooses not to subdivide the image, and codes all of

the coefficients with one quantization strategy. Good examples of this can be seen in the images Mandrill and Bridge, shown in figure 2.25. There is no large section of either image that is completely smooth. All parts of the images are detailed, as can be seen by the large values in Tables 2.17 and 2.18.



(a) (b)

Figure 2.25: Test Images: (a) Mandrill (b) Bridge

0.99236	0.99166
0.98886	0.98980

Table 2.17: Kirsch18 values for the four quadrants of Mandrill

0.98929	0.99181
0.99250	0.98816

Table 2.18: Kirsch18 values for the four quadrants of Bridge

These Kirsch18 values call for ModLVQ's vector quantization to be used in every quadrant, and thus no quadtree subdivision is performed. Each image is coded in its entirety using the vector quantization. The results are presented in Tables 2.19 and 2.20.

In every case, Hy-Q chooses the better quantization strategy, ModLVQ. It may be noted that in a few cases, Hy-Q shows a very slight improvement even over ModLVQ. This is due to an observation made by Knipe in [Kni96]. He discovered that in certain situations, the SPIHT algorithm codes some redundant information concerning the descendants of coefficients in the coefficient hierarchy. This redundancy was removed in the implementation of ModLVQ, and thus appears in my hybrid version, Hy-Q.

Bits Per Pixel	SPIHT	ModLVQ	Hy-Q	Best Result
1.0	29.17	29.42	29.42	Hy-Q
0.5	25.64	25.97	25.97	Hy-Q
0.25	23.27	23.59	23.59	Hy-Q
0.125	21.72	21.91	21.92	Hy-Q
0.0625	20.73	20.81	20.81	Hy-Q

Table 2.19: PSNR results for Mandrill

Bits Per Pixel	SPIHT	ModLVQ	Hy-Q	Best Result
1.0	29.48	29.79	29.79	Hy-Q
0.5	26.39	26.60	26.61	Hy-Q
0.25	24.33	24.47	24.48	Hy-Q
0.125	22.66	22.84	22.85	Hy-Q
0.0625	21.25	21.41	21.43	Hy-Q

Table 2.20: PSNR results for Bridge

The same results can be seen on images that are mostly smooth. Consider, for instance, the natural image Bird and the synthetic image Quad-lzw, seen in Figure 2.26. Each has large regions that feature little pixel variation. The results of measuring the quadrants, as seen in Tables 2.21 and 2.22, confirm that SPIHT is the correct quantization strategy to use for each image. The PSNR results are shown in Tables 2.23 and 2.24.

0.72153	0.80852
0.78616	0.88486

Table 2.21: Kirsch18 values for the four quadrants of Bird

0.28598	0.30069
0.32783	0.33469

Table 2.22: Kirsch18 values for the four quadrants of Quad-lzw

Once again, Hy-Q chooses the better strategy for every bpp value. As was the case with detailed images, Hy-Q sometimes shows an improvement in results even over those of SPIHT.

2.4.2 Mixed Images

In order to determine the success of Hy-Q's approach to quantization, it is most important to examine the results on images that combine both detail and smooth-

Bits Per Pixel	ModLVQ	SPIHT	Hy-Q	Best Result
1.0	44.42	44.56	44.57	Hy-Q
0.5	40.91	41.35	41.36	Hy-Q
0.25	36.99	37.77	37.78	Hy-Q
0.125	33.45	34.08	34.11	Hy-Q
0.0625	30.05	30.58	30.64	Hy-Q

Table 2.23: PSNR results for Bird

Bits Per Pixel	ModLVQ	SPIHT	Hy-Q	Best Result
1.0	50.72	54.15	54.15	Hy-Q
0.5	41.18	43.78	43.79	Hy-Q
0.25	34.89	36.79	36.80	Hy-Q
0.125	30.60	31.67	31.67	Hy-Q
0.0625	27.12	27.98	27.98	Hy-Q

Table 2.24: PSNR results for Quad-lzw

(a) (b)

Figure 2.26: Test Images: (a) Bird (b) Quad-lzw

ness. These are the images that will be subdivided, using a quadtree, and quantized using a mixture of scalar and vector quantization. If results can be obtained that are better than those from using only one quantization strategy, then the approach taken by the algorithm can be justified as a worthwhile one. This is, indeed, the case.

Hy-Q was found to work best on images in which the quadrants are all either mostly smooth or mostly detailed. This is an intuitive observation, based on the current approach that Hy-Q takes, although it does not work perfectly, due to

the nature of the quantization process. For instance, a mostly detailed quadrant might not be compressed better with ModLVQ due to holes in the vector codebook. Nonetheless, the observation works well enough that a small number of images from the testing set that would ordinarily have been subdivided by Hy-Q were compressed using only one quantization strategy, using the manual override capabilities of Hy-Q.

Judging which images will work well with Hy-Q and which will not can easily be done by the human eye. It was found during the testing phase of Hy-Q that after only an hour of practice, a human operator could determine correctly whether or not an image would be suited to Hy-Q well over 80% of the time.

Consider the image Shepherd, seen in Figure 2.27. The image is a drawing, with a man in the foreground, and a plain and hills in the background. The results of applying Hy-Q's Kirsch18 measure to the image are shown in Table 2.25. As can be seen in the table, for this image, two levels of quadtree subdivision are required.

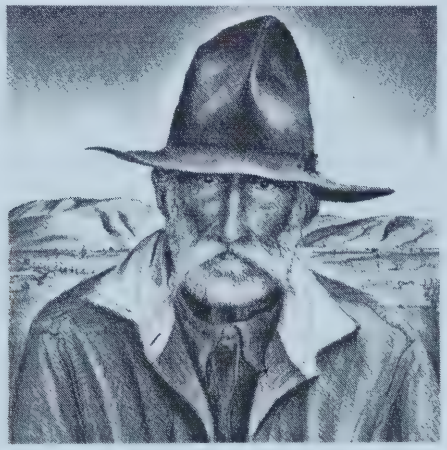


Figure 2.27: Test Image: Shepherd

0.98828	0.97392	0.89928	0.97531
0.44331	0.84946	0.92819	0.56135
0.96699	0.93670	0.89979	0.97285
0.98639	0.98917	0.93115	0.99295

Table 2.25: Kirsch18 values for Shepherd (2 levels of quadtree decomposition)

The actual subdivision, shown in Figure 2.28, confirms what our eyes would suggest to be reasonable. The areas of sky to the left and right of the shepherd's head are particularly smooth, and even the top of the shepherd's hat is fairly uniformly dark. There are two other sections of the image to be coded by SPIHT, and the ones that contain most of the shepherd's jacket lapels. These features are also mostly uniform in intensity. The rest of the image shows a moderate amount of detail,

largely due to the way in which it was drawn.

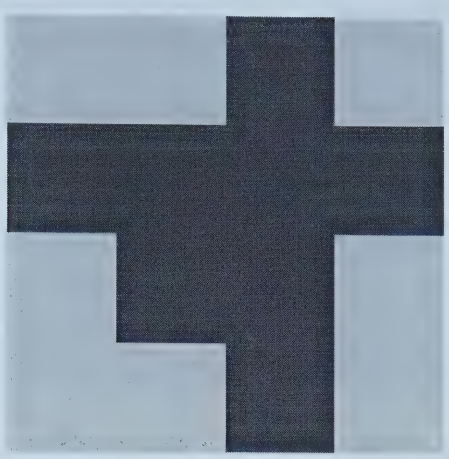


Figure 2.28: 2-level quadtree subdivision of Shepherd (light grey = ModLVQ, black = SPIHT)

The results of compressing Shepherd using Hy-Q are presented in Table 2.26. For every bpp value, the PSNR result of Hy-Q equals or surpasses that of both SPIHT and ModLVQ.

Bits Per Pixel	ModLVQ	SPIHT	Hy-Q	Best Result
1.0	32.23	32.24	32.41	Hy-Q
0.5	29.08	29.03	29.15	Hy-Q
0.25	27.00	27.00	27.05	Hy-Q
0.125	25.40	25.48	25.49	Hy-Q
0.0625	23.88	24.03	24.03	Hy-Q

Table 2.26: PSNR results for Shepherd

Another image on which Hy-Q takes advantage of quadtree subdivision is Bay, seen in Figure 2.29. The bottom half of the image shows some water and trees, and is reasonably detailed. The top half of the image is quite smooth, featuring a cliff and the sky. The visual assessment of this image is confirmed by the Kirsch18 results, shown in Table 2.27, and the quadtree subdivision, shown in Figure 2.30.

0.88416	0.84933
0.98910	0.98936

Table 2.27: Kirsch18 values for the four quadrants of Bay

The PSNR results for Bay are good, and are shown in Table 2.28. For every bpp value, Hy-Q surpasses both SPIHT and ModLVQ.

Figure 2.29: Test Image: Bay

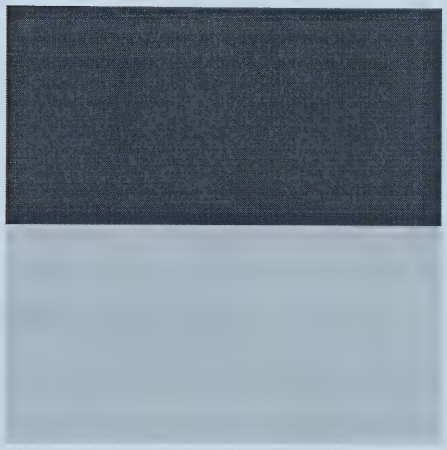


Figure 2.30: 1-level quadtree decomposition of Bay

Bits Per Pixel	ModLVQ	SPIHT	Hy-Q	Best Result
1.0	33.77	33.72	33.87	Hy-Q
0.5	29.99	30.00	30.14	Hy-Q
0.25	27.63	27.56	27.68	Hy-Q
0.125	26.00	25.95	26.03	Hy-Q
0.0625	24.78	24.62	24.79	Hy-Q

Table 2.28: PSNR results for Bay

2.4.3 Some Images On Which the Algorithm Doesn't Work Well

In the interests of giving the full picture of the results achieved by Hy-Q, this section will show two examples of the images on which Hy-Q does not achieve results superior to those of SPIHT and ModLVQ.

The first image is the image House, shown in Figure 2.13. The top half and bottom left quadrant of the image are dominated by the smooth sky, while the bottom right quadrant of the image contains mostly the detail of the house itself. The Kirsch18 values for the quadrants of House are shown in Table 2.12. A second level of quadtree decomposition is possible, but does not yield better empirical results.

The compression results for House are shown in Table 2.29. At 1 bpp, Hy-Q significantly outperforms both SPIHT and ModLVQ. Below this value, however, SPIHT takes over and provides the best results. It should be noted that at every bpp value, Hy-Q's results are better than those of ModLVQ.

Bits Per Pixel	ModLVQ	SPIHT	Hy-Q	Best Result
1.0	39.17	39.21	39.40	Hy-Q
0.5	34.82	35.15	35.00	SPIHT
0.25	31.63	31.99	31.86	SPIHT
0.125	28.46	29.00	28.76	SPIHT
0.0625	25.73	25.97	25.84	SPIHT

Table 2.29: PSNR results for House

The second image presented in this section is Lax, seen in Figure 2.31. It is difficult to tell from looking at the image whether or not it should be classified as smooth or detailed. Some sections of the image are smooth, others are detailed, but no individual quadrant is completely one or the other. Kirsch18 produces the results shown in Table 2.30. All of the values are above the threshold, and thus the entire image is coded using ModLVQ's vector quantization.

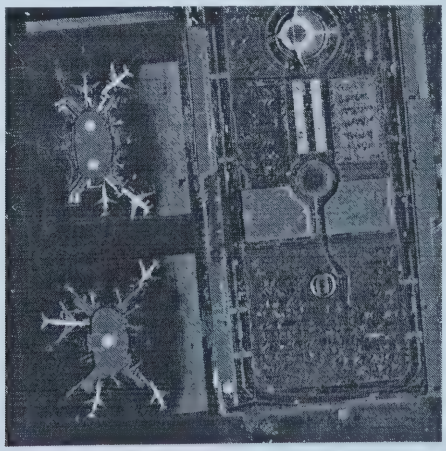


Figure 2.31: Test Image: Lax

As the results (Table 2.31) show, however, SPIHT's scalar quantization would

0.97628	0.99161
0.98975	0.96302

Table 2.30: Kirsch18 values for the four quadrants of Lax

likely have been the better choice. For every bpp value except 0.125, SPIHT produces results that are better than ModLVQ's.

Bits Per Pixel	ModLVQ	SPIHT	Hy-Q	Best Result
1.0	30.42	30.67	30.42	SPIHT
0.5	27.35	27.54	27.35	SPIHT
0.25	25.12	25.16	25.12	SPIHT
0.125	23.52	23.46	23.52	Hy-Q
0.0625	22.40	22.42	22.41	SPIHT

Table 2.31: PSNR results for Lax

2.4.4 Results Summary

Hy-Q was tested on 70 widely different images, ranging from natural to synthetic images, indoor to outdoor scenes, etc. For each of these images, five different bitrates were tested.

Hy-Q decided to use only one quantization strategy (i.e. no quadtree subdivision) on 43 of the images. SPIHT was the strategy used for 25 of these images, while the other 18 were quantized using ModLVQ. In these 215 separate tests, the results of Hy-Q were equal to, or even a little better than, both SPIHT and MODLVQ 85.58% of the time.

The other 27 images from the testing set fit the criterion for ideal Hy-Q compression, namely that their quadrants are either mostly smooth or mostly detailed. Of the 135 separate tests run on this set of images, Hy-Q's results were equal to or better than those of both SPIHT and MODLVQ 77.04% of the time.

These results are summarized in Tables 2.32 and 2.33.

Bits Per Pixel	Images For Which Hy-Q's Results Are Best	Percentage
1.0	36	84%
0.5	35	81%
0.25	37	86%
0.125	39	91%
0.0625	37	86%
Overall		**85.58%**

Table 2.32: Overall Results for Hy-Q on Images Not Quadtree-Subdivided

	Compared To SPIHT and ModLVQ			
Bits Per Pixel	Worse	Equal	Better	Percentage Equal or Better
1.0	9	2	16	67%
0.5	4	1	22	85%
0.25	7	2	18	74%
0.125	4	4	19	85%
0.0625	7	5	15	74%
Overall				**77.04%**

Table 2.33: Overall Results for Hy-Q on Images That Are Quadtree-Subdivided

2.5 Future Work

The most important future direction for work on Hy-Q is to apply the framework to different quantization strategies than SPIHT's scalar and ModLVQ's vector quantization. In the time since ModLVQ was first introduced, various new quantization strategies have been proposed, among them [HM96], [FA97] and [HGS97]. The most promising of these could be analysed to determine on what types of images or coefficients they give the best results. This information could then be used, in conjunction with the Hy-Q framework, to produce even better results, and further improve the quality of compressed images.

There is also room for further research into the framework itself. The following is a list of possible directions for investigation:

- Look into new ways of creating a measure for determining which quantization strategy should be used. A method that overcomes the difficulties discussed in Section 2.2.2, and works directly on the wavelet coefficients, instead of the original pixels of the image, and takes into account the desired compression ratio, would be of great service. Not only would it most likely lead to better results from Hy-Q, allowing it to make more accurate selections concerning which quantization algorithm to use, but it would also lend important insight into how future measures should be designed.

- Examine different image segmentation methods. The quadtree method used in Hy-Q has the benefits of being efficient to code, and corresponding well with the parent-child relationship between coefficient blocks. However, it suffers from being too rigid, perhaps ignoring a large detailed region that falls along the quadtree division boundaries. The tradeoffs between increased coding cost, in terms of space and time, and improved image segmentation could be investigated. First, though, the difficulties described in Section 2.3.1 would have to be overcome.

2.6 Conclusion

This chapter presents a new contribution to the literature on wavelet image compression. The proposed new method mixes two quantization strategies within one compression framework, even though different block sizes are used by each type of quantization. The method, which I have named Hy-Q, achieves better compression results on most images than either compression strategy used by itself can offer.

The main contribution to the literature of this new method, however, is the idea itself of mixing seemingly incompatible quantization strategies. The power of this idea may allow future quantization strategies to be mixed in a similar fashion to Hy-Q's approach. Thus, even though the results produced by Hy-Q may be far surpassed by newer methods, the idea behind Hy-Q can still be used to achieve even better results.

Chapter 3

Implementing the Dual Wavelet Frame Transform

One of the new directions in wavelet research involves investigating wavelet frames, which are "associated with oversampling, or redundancy" ([SN97]). Mathematically, this means that a frame has extra vectors beyond what are required for a basis. Practically, this means that the wavelet transforms apply more than one highpass filter.

Because of the extra highpass filter(s), and thus extra wavelet coefficients, image compression using frame transforms is more difficult than with standard orthogonal or biorthogonal wavelet filters. The motivation behind using frames is that these extra data can be offset through some desirable properties designed into the extra filters.

Solving all of the issues associated with using the frame transforms for image compression is well beyond the scope of this thesis. The goal of the project described in this chapter was to implement a special type of frame transform, the dual frame transform. The implementation of this type of frame has received no attention in the literature so far, and thus some practical issues arising out of the implementation will be discussed.

This chapter is organized as follows: Section 3.1 gives a brief, high-level overview of the theory behind dual frames. Section 3.2 discusses some issues associated with storing the coefficients generated by the dual frame transform. Section 3.3 reviews some issues associated with applying filters at the boundaries of the data. Sections 3.4 and 3.5 discuss some of the difficulties that arise from the θ filter associated with dual frames. Section 3.6 gives some basic results of the dual frame transform applied to denoising, which is a simpler problem than compression. Finally, Section 3.7 describes directions for future research on the dual frame transform, especially with respect to image compression.

3.1 Background

A very simple way in which to explain a frame is to describe it as an overcomplete representation of a particular space. For instance, the three vectors shown in Figure 3.1 comprise a frame in $\mathbb{R}^2$.

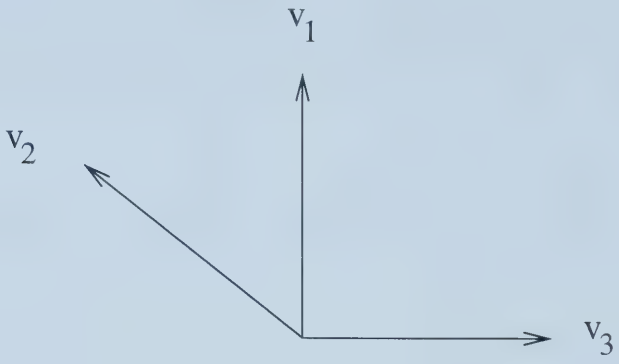


Figure 3.1: A Frame for $\mathbb{R}^2$

The following is the definition of a frame. If we take a set of elements $\{h_k\}_{k\in\mathbb{Z}}$ in some Hilbert space H such as $\mathbb{R}^2$, then the following must be true for this set to be a frame:

$$\exists A, B > 0 \quad \text{st} \quad A\|t\|^2 \leq \sum_{k\in\mathbb{Z}} |\langle t, h_k\rangle|^2 \leq B\|t\|^2 \qquad \forall t \in H \tag{3.1}$$

Because of the redundancy of a frame, any element in H can be represented in many different ways. For instance, if we want to represent an element $f \in H$ as $\sum_{k\in\mathbb{Z}} c_k h_k$, the sequence c_k is not unique.

We are interested in wavelet frames, of which there are two main types. The first is called a **tight wavelet frame**, which is a generalization of an orthonormal wavelet basis. In Equation 3.1 above, the condition that $A = B$ must be met in order for a frame to be considered tight. Another distinguishing feature of a tight wavelet frame is that it can be generated by a single family of related functions. In the context of image compression, this means that both the decomposition and the reconstruction filters are generated from the same set of functions. This type of wavelet frame has received a large amount of attention in the literature.

The second type of wavelet frame is called a **dual wavelet frame**, which is a generalization of biorthogonal wavelets. As the name suggests, dual wavelet frames are generated by two separate families of functions. Whereas in a tight wavelet frame, the reconstruction and decomposition filters are strongly related, there is more freedom in the design of dual wavelet filters. Part of this freedom comes from the introduction of an extra filter, denoted by θ.

The role of θ can be seen by examining the perfect reconstruction formula for dual wavelet frames. In this formula, the filters are written in z-transform notation. This means that if we have a filter represented by the sequence h_n, we can rewrite this as $h(z) = \sum_n h_n z^n$. Also, h represents lowpass filters, and g represents highpass filters. Note that in theory, any number of highpass filters can be used. For the purposes of image compression, however, using more than two highpass filters would generate far too much data. Thus, the following formula covers the specific case of two highpass filters.

$$\begin{bmatrix} h(z) & zg^1(z) & zg^2(z) \\ h(-z) & (-z)g^1(-z) & (-z)g^2(-z) \end{bmatrix} \begin{bmatrix} \theta(z^2) & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \overline{\tilde{h}(z)} & \overline{\tilde{h}(-z)} \\ \frac{1}{z}\overline{\tilde{g}^1(z)} & -\frac{1}{z}\overline{\tilde{g}^1(-z)} \\ \frac{1}{z}\overline{\tilde{g}^2(z)} & -\frac{1}{z}\overline{\tilde{g}^2(-z)} \end{bmatrix}$$

$$= \begin{bmatrix} 2\theta(z) & 0 \\ 0 & 2 \end{bmatrix}$$

It may be noted from examining the perfect reconstruction formula that biorthogonal wavelets are in fact a special case of dual wavelet frames. If we set $\theta = 1, g^2 = 0$ and $\tilde{g_2} = 0$, we have the perfect reconstruction formula for biorthogonal wavelets. The extra highpass filter(s), along with θ, give a great deal of freedom to the designers of dual wavelet frame filters, easily allowing them to tailor the properties of the filters to the specific tasks for which they are being created.

In the practical implementation of dual wavelet frame transforms, the theta appears at the end of the decomposition, as well as at the end of the reconstruction. Once the last level of decomposition has been performed, convolution is performed between θ and the remaining coarse lowpass coefficients. After the last level of reconstruction, this convolution is "undone" by performing deconvolution with the θ filter.

Figure 3.2 shows a block diagram of one level of the dual wavelet frame transform.

3.2 Space Issues

Because of the extra highpass filter(s) employed in the dual frame transforms, the amount of space that is required for coefficients takes on great importance. This section will serve two purposes. The first is to describe how the data structure used to store coefficients in SPIHT can be modified to handle the extra coefficients. The other is to discuss the space-related difficulties faced by a compression scheme using the dual frame transform.

The regular wavelet transform is impressively space-efficient. If an $n \times n$ image is to be compressed, exactly $n \times n$ coefficients are produced by the transform. At each stage of the transform, the current LL_n area of coefficients is replaced by

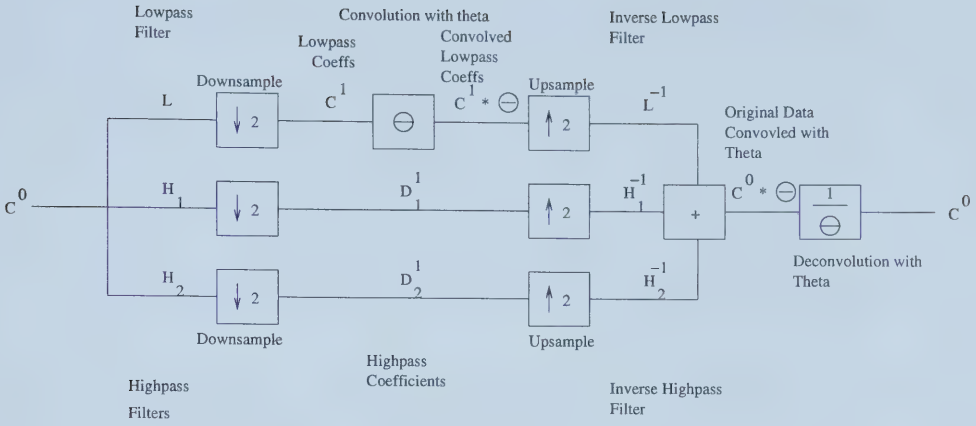


Figure 3.2: One Level of the Dual Wavelet Frame Transform

four new groups of coefficients, LL_{n+1}, HL_{n+1}, LH_{n+1} and HH_{n+1}. Each of these subbands is one quarter the size of LL_n, and thus no additional storage is required to accommodate the new level of transform. This can be seen in Figure 2.3.

For the case of the dual frame transform with two highpass filters, this scheme will not work. The tensor product, when applied with three one-dimensional filters, will produce 9 new subbands, each of which is one quarter the size of the transformed LL_n subband. The extra subbands would overwrite the coefficients from the previous transform level if stored in the way described above.

Because of the fact that this thesis deals with image compression, a solution to this problem that accommodates standard compression methods, notably SPIHT, was sought. The most important feature of SPIHT is its prediction of (in)significant coefficients between subbands, which requires knowledge of how coefficients are descended from each other.

In the case of the standard wavelet transform, this is a simple operation. A coefficient located at (x, y) will have four descendants at coordinates $(2x, 2y)$, $(2x + 1, 2y)$, $(2x, 2y + 1)$ and $(2x + 1, 2y + 1)$. This can be easily verified by examining the relative positions of the subbands in the wavelet decomposition. A storage scheme that preserves this simplicity in descendant calculation would be ideal.

The solution that has been introduced into SPIHT is to store each level of transform in a separate two-dimensional array. Each level of the pyramid stores one resolution of the dual frame transform coefficients, and can be used for temporary storage during the transform process. Conceptually, this creates a true wavelet "pyramid", as each level of transform produces fewer coefficients than the last, and thus the size of the two-dimensional arrays decreases with each level. This is illustrated in Figure 3.3.

Each coefficient preserves its x and y coordinate, and its z coordinate is simply

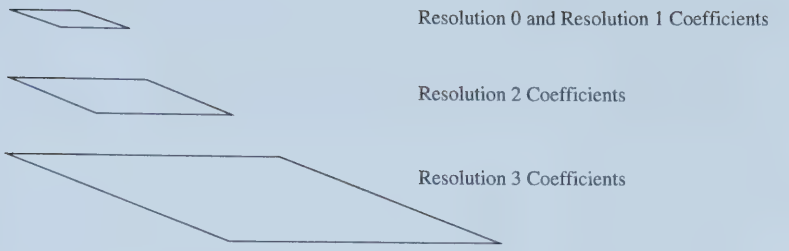


Figure 3.3: Data Structure Used in SPIHT for Storing Frame Coefficients

the level of transform at which it was generated. Descendants of a coefficient can be found by doubling the x and y coordinates of the parent, as described above, and simply decreasing the level by 1. This is done because the descendants of a coefficient appear in the next lowest resolution subband, and will thus be stored on the next lowest level of the storage pyramid.

Table 3.1 illustrates the amount of storage required by this scheme. For this example, a 256×256 image is the input, and 5 levels of transform are applied.

Coefficient Type	Wavelet Coeffs	Frame Coeffs	Space In New Scheme
Resolution 0	$3 * 128^2$	$8 * 128^2$	384^2
Resolution 1	$3 * 64^2$	$8 * 64^2$	192^2
Resolution 2	$3 * 32^2$	$8 * 32^2$	96^2
Resolution 3	$3 * 16^2$	$8 * 16^2$	48^2
Resoultion 4	$3 * 8^2$	$8 * 8^2$	24^2
Resolution 5	$4 * 4^2$	$9 * 4^2$	12^2
Total	65536	174736	196560

Table 3.1: Example storage requirements of the dual frame transform

As can be seen in the table, the efficiency in descendant calculation is offset by some wasted storage (over 200 KB). For the purposes of compression, however, computation time is at a far greater premium than temporary storage space, and thus this is a worthwhile tradeoff. If the space is an issue, more space-efficient storage schemes can easily be designed.

The redundancy of the frame transform with two highpass filters over the regular wavelet transform is a little less than 8/3 (in the above example, it's 2.66626). This poses a large challenge for compression. For the same bit budget, well over twice the coefficients have to be coded. Solving these issues is beyond the scope of this thesis, but the following are some possibilities:

- Having extra highpass filters provides redundancy that can be exploited in the design of the filters.
- The highpass filters are often similar (e.g. shifted versions of each other). Some

prediction can be done between the highpass subbands to save coefficients.

- Two-dimensional frame filters can be designed to reduce the redundancy factor to a ratio of 5/4, instead of 8/3.

- The option of combining a dual frame filter with a regular orthogonal or biorthogonal wavelet filter in a tensor product can be investigated. In other words, the rows could be transformed with the dual frame filter, and the columns with the regular wavelet filter. This would reduce the redundancy to a factor of 5/3.

3.3 Handling the Boundary Case

One of the issues that needs to be dealt with by any wavelet transform is how to handle the boundary cases, when the wavelet filter is centred at the beginning or end of the data array. This topic is quite well understood, but is often considered to be inconsequential in the literature, and is not discussed in much detail. Unless one has tried to implement the wavelet transform, however, all of the issues might not be perfectly clear. The following gives a detailed overview of the issues involved in handling the boundary case. Note that only filters of odd length were investigated for this work. Similar rules apply to filters of even length.

Consider the case of transforming an image. When the wavelet filter is centred at the beginning or the end of a row or column of pixels, part of the filter will "hang off the edge" of the array.

This can also be seen from the mathematical point of view. The filtering process involves discrete convolution. The formula for this type of convolution is as follows, where d represents the data array, a represents the wavelet filter, and c represents the output coefficients:

$$c_n = \sum_k a_k d_{n-k} \tag{3.2}$$

When calculating the first few coefficients, the value of n is small, and thus when k is larger than n, the index into the data array will be negative. This index is outside the domain of the data, so the designer of a program involving wavelet transforms is free to put in any value at that index. Here is a list of the strategies that are in common use:

- End Value (EV) - copy the end value of the data array as many times as necessary to match up with the filter

- Zero Padding (ZP) - simply pad the end of the data array with zeros

- Periodicity (P) - copy values from the other end of the array
- Reflection (R) - use the end value as a mirror, reflecting values around it. Note that two different possibilities exist in this category, namely repeating the end value (RR) and not doing so (RN).

Table 3.2 illustrates each of these strategies on a data array $d_0 - d_7$.

EV	d_0	d_0	d_0	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_7	d_7
ZP	0	0	d_0	d_1	d_2	d_3	d_4	d_5	d_6	d_7	0	0
P	d_6	d_7	d_0	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_0	d_1
RR	d_1	d_0	d_0	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_7	d_6
RN	d_2	d_1	d_0	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_6	d_5

Table 3.2: Illustrations of various boundary strategies

Each of these strategies is theoretically sound, and, if implemented correctly, will allow for perfect reconstruction. For the purposes of implementing the dual frame transform, however, reflection was the preferred strategy. The reason lies in the fact that the coefficients generated by the forward frame transform should give an accurate reflection of local pixel intensity changes. Padding with zeros creates artificially small values, while using the end value may negate some high frequencies at the boundaries. Both of these strategies require that extra coefficients beyond the boundaries be kept for the inverse transform. Periodicity copies whole sections of the image intact, but the frequency at one side of the image may be completely different from the frequency at the other side. Reflection overcomes each of these difficulties.

Reflection must be done for both the forward and the inverse transforms. The key issue is ensuring that the reflection strategy chosen for the forward transform generates wavelet coefficients that can be reflected correctly before the inverse transform. In order to prove that this can be done, it is necessary to examine the coefficients generated when the wavelet filters are centred outside the boundaries of the data array.

Consider an example in which a data array of length 8 $[d_0 - d_7]$ is being transformed using a symmetric filter of length 3 ($[a_1 a_0 a_1]$), using the boundary strategy of reflection without repeating the end value. Because of downsampling, only 4 coefficients are kept, and thus the filter is applied with its centre at d_0, d_2, d_4 and d_6. This is shown in Table 3.3. The notation used in this diagram is that each time the filter, which acts as a mask, is applied, its coefficients are written above the data values with which they will be multiplied (or alternately below, to avoid overlapping).

The four wavelet coefficients generated by this process are given here:

Filter:		a_1	a_0	a_1		a_1	a_0	a_1				
Data:	d_2	d_1	d_0	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_6	d_5
Filter:				a_1	a_0	a_1		a_1	a_0	a_1		
Coefficient:			c_0		c_1		c_2		c_3			

Table 3.3: Applying a wavelet filter to a data array

$$c_0 = a_1 * d_1 + a_0 * d_0 + a_1 * d_1$$
$$c_1 = a_1 * d_1 + a_0 * d_2 + a_1 * d_3$$
$$c_2 = a_1 * d_3 + a_0 * d_4 + a_1 * d_5$$
$$c_3 = a_1 * d_5 + a_0 * d_6 + a_1 * d_7$$

The coefficients beyond the boundary of this array (e.g. c_{-1}, c_{-2}, etc.), can be calculated simply by moving the forward transform filter beyond the boundaries of the original data array, with a sufficient number of coefficients reflected about each end. The following are the calculations of these "external" coefficients:

$$c_{-2} = a_1 * d_5 + a_0 * d_4 + a_1 * d_3 = c_2$$
$$c_{-1} = a_1 * d_3 + a_0 * d_2 + a_1 * d_1 = c_1$$
$$c_4 = a_1 * d_7 + a_0 * d_6 + a_1 * d_5 = c_3$$
$$c_5 = a_1 * d_5 + a_0 * d_4 + a_1 * d_3 = c_2$$

The array of coefficients after the forward transform looks as follows:

c_2	c_1	c_0	c_1	c_2	c_3	c_3	c_2

Table 3.4: Coefficient array after forward transform

Thus, the inverse transform can reflect about the left side of the array without repeating the end value, and reflect about the right side of the array with the end value repeated, and have all of the necessary coefficients available.

The following set of equations, along with Table 3.5, illustrates a case in which the forward transform does not provide proper reflection about the boundaries of the coefficients. It is the same case as above, but for the forward transform, the end value is repeated.

Filter:		a_1	a_0	a_1		a_1	a_0	a_1				
Data:	d_1	d_0	d_0	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_7	d_6
Filter:				a_1	a_0	a_1		a_1	a_0	a_1		
Coefficient:			c_0		c_1		c_2		c_3			

Table 3.5: Forward transform, using reflection with the end value repeated

$$
\begin{aligned}
c_{-2} &= a_1 * d_4 + a_0 * d_3 + a_1 * d_2 \\
c_{-1} &= a_1 * d_2 + a_0 * d_1 + a_1 * d_0 \\
c_0 &= a_1 * d_0 + a_0 * d_0 + a_1 * d_1 \\
c_1 &= a_1 * d_1 + a_0 * d_2 + a_1 * d_3 \\
c_2 &= a_1 * d_3 + a_0 * d_4 + a_1 * d_5 \\
c_3 &= a_1 * d_5 + a_0 * d_6 + a_1 * d_7 \\
c_4 &= a_1 * d_7 + a_0 * d_7 + a_1 * d_6 \\
c_5 &= a_1 * d_6 + a_0 * d_5 + a_1 * d_4
\end{aligned}
$$

The "external" coefficients c_{-2}, c_{-1}, c_4 and c_5 are different from $c_0 - c_3$. The result is that if the inverse transform is going to be able to reconstruct the original data array properly, it will have to be explicitly given the values of the extra coefficients, as reflection is impossible. For compression, this means extra bits are used when they aren't required.

3.3.1 Different Cases

Filters with Different Symmetry Centres For the filter used in the example above, the symmetry centre is at 0. This means that when the filter is applied at the beginning of the data array, the centre of the filter lines up with the first data value. For some filters, including many highpass wavelet filters, the symmetry centre is at 1. This results in the following arrangement:

Filter:			a_1	a_0	a_1		a_1	a_0	a_1			
Data:	d_2	d_1	d_0	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_6	d_5
Filter:					a_1	a_0	a_1		a_1	a_0	a_1	
Coefficient:				c_0		c_1		c_2		c_3		

Table 3.6: Applying a wavelet filter to a data array

This case is very similar to the first case presented above. In fact, since the filter is symmetrical, the same argument can be applied, but with everything reversed.

In other words, take the example above, and flip it around so that the left side is the right, and vice-versa. The result is a picture that looks exactly like Table 3.6. The solution to this case, then, is still to reflect without repeating the end value for the forward transform. For the inverse transform, the right side of the coefficient array should be reflected with the end value repeated, and the left side should be reflected without repeating the end value.

Asymmetrical Filters Some filters are not symmetrical. In general, reflection may not be the best choice for these cases, unless a special condition is met, namely that one half of the filter be the negative of the other half, and the middle value be 0. The filter $[a_1 0 - a_1]$ is one such example. Assume for the example that the symmetry centre of this filter is at 0. Table 3.7 shows an example of such a filter.

Filter Type	-1	0	-1
Forward Lowpass	1/4	1/2	1/4
Forward Highpass 1	$-1/4$	1/2	$-1/4$
Forward Highpass 2	$-\sqrt{2}/4$	0	$\sqrt{2}/4$
Inverse Lowpass	1/4	1/2	1/4
Inverse Highpass 1	$-1/4$	1/2	$-1/4$
Inverse Highpass 2	$-\sqrt{2}/4$	0	$\sqrt{2}/4$

Table 3.7: Frame Filter Generated From B-Spline Function of Order 2

The following are the calculations from this case:

$$
\begin{aligned}
c_{-2} &= a_1 * d_5 + 0 * d_4 + -a_1 * d_3 = -c_2 \\
c_{-1} &= a_1 * d_3 + 0 * d_2 + -a_1 * d_1 = -c_1 \\
c_0 &= a_1 * d_1 + 0 * d_0 + -a_1 * d_1 \\
c_1 &= a_1 * d_1 + 0 * d_2 + -a_1 * d_3 \\
c_2 &= a_1 * d_3 + 0 * d_4 + -a_1 * d_5 \\
c_3 &= a_1 * d_5 + 0 * d_6 + -a_1 * d_7 \\
c_4 &= a_1 * d_7 + 0 * d_6 + -a_1 * d_5 = -c_3 \\
c_5 &= a_1 * d_5 + 0 * d_4 + -a_1 * d_3 = -c_2
\end{aligned}
$$

$-c_2$	$-c_1$	c_0	c_1	c_2	c_3	$-c_3$	$-c_2$

Table 3.8: Coefficient array after transform

Table 3.8 shows that the reflection strategy for the inverse transform in this case is to reflect as usual, but to negate the reflected values.

3.4 Performing Deconvolution

The wavelet frame transform is performed in exactly the same way as the regular wavelet transform, with two differences. The obvious difference is that instead of one highpass filter, there are two (or more). One way to deal with this issue was discussed in Section 3.2. The other difference is that at the last step of both the forward and the inverse transforms, convolution must be done with the filter θ.

In the forward direction, there is not much of a problem. Convolution is performed between the highest resolution set of coarse coefficients and the θ filter. Since several levels of transform are usually applied, the number of coarse coefficients is typically small, and the θ filter is compactly supported, usually no longer than the frame filters themselves. Thus the convolution is not computationally expensive. As can be seen from Equation 3.2, the length of the data output by the convolution operator is longer than the input data array, but the extra coefficients can be accounted for by reflection, and thus no additional storage is required as a result of the convolution.

$$C^{n-1} * \theta = L^{-1} * (C^n * \theta) + H_1^{-1} * D^{n,1} + H_2^{-1} * D^{n,2} \tag{3.3}$$

When it comes time to reverse the effects of the convolution, however, a problem arises in the implementation. After the data have been reconstructed, following the dual frame reconstruction formula, which can be seen in Equation 3.3, we have the original number of data values, but they are still under the influence of the convolution with θ. In other words, we have the following:

$$C^0 * \theta$$

From the theoretical standpoint, arriving back at the original data involves deconvolution with θ, which is the same as performing convolution with the inverse of θ. The derivation is as follows:

$$\begin{aligned} (C^0 * \theta) * 1/\theta &= C^0 * (\theta * 1/\theta) \\ &= C^0 * 1 \\ &= C^0 \end{aligned}$$

This is made possible by the following well-known facts:

- the convolution operation is associative
- the convolution of a function with its inverse equals 1

Unfortunately, the convolution with $1/\theta$ causes problems. Because of the fact that θ is a compactly supported function, its inverse will have infinite support. Translating this to the case of discrete convolution, this would require a filter of infinite length, which is obviously impossible.

In order to deal with this problem, a tradeoff must be made. The length of the inverse θ filter can be reduced by using an approximation to the infinite sequence. This necessarily causes a reduction in the accuracy of the reconstruction of the original data values. If a balance can be struck between the length of the approximation and the error in the reconstruction, the problem can be solved in practice. Experimentation showed, however, that even using up to 127 terms in the approximation to the theta inverse filter gave results that produced too much error for this approach to be used in applications.

Alternate solutions are available, however. One such solution is to "simulate" the convolution process, determining what original data array, when convolution with theta is performed on it, would give us our current data. Since the data array is equivalent to a vector, the convolution operation can easily be viewed as the multiplication of a matrix and the data vector. Therefore, performing the deconvolution is the same as solving the linear system $Ax = b$, where A is the convolution matrix, x is C^0, the data we are looking for, and b is $C^0 * \theta$, the data we have.

Many different methods exist for solving linear systems. The one chosen for the frame transform is LU decomposition, documented in many places, including [Hea97]. This process decomposes the matrix A into the product of a lower triangular matrix L and an upper triangular matrix U. Our system now looks as follows:

$$LUx = b$$

If we make the substitution $Ux = y$, the system becomes $Ly = b$. Because of the fact that L is lower-triangular, solving this system is a simple matter of performing forward-substitution. The final step is to solve for x by performing back-substitution on the system $Ux = y$.

LU decomposition is a suitable choice for this situation because of the fact that this system will have to be solved many times, once for each row and column in the image. For each case, only the array b changes. The matrix A remains constant, since it represents the convolution process, which depends only on θ. Thus, the LU decomposition needs to be done only once, giving a relatively efficient implementation.

3.5 Implementation Difficulties

As often happens, the transition from theory to practice is not perfectly smooth. There is a problem with performing multiple levels of the frame transform *when* $\theta \neq 1$. The problem arises with the lowpass coefficients and reflection.

In order to see the problem, consider the following example. Tables 3.9 and 3.10 show the filter values (rounded to four decimal places), along with their alignment, for a Spline frame filter. This filter was constructed using the algorithm described in [DH00], and the following two lowpass filters:

$$h(x) = (1+x)^4/8$$
$$\tilde{h}(x) = (1+x)^4/8$$

Index	Lowpass	Highpass 1	Highpass 2
-3	0.0	0.0	0.0
-2	0.125	1.0	0.0
-1	0.5	−4.0	1.0
0	0.75	6.0	−4.0
1	0.5	−4.0	6.0
2	0.1250	1.0	−4.0
3	0.0	0.0	1.0

Table 3.9: Forward Filter Values for a Spline Frame Filter

As the table shows, the forward lowpass filter, as is most often the case, has its symmetry centre at 0. The result is that after the first level of transform, the resulting lowpass coefficients at the right end of the array have repeating symmetry.

If another level of transform is performed, however, the problem arises. The only pre-transform reflection strategy that allows for prediction of the coefficients beyond the boundary of the post-transform coefficients is reflection *without repeating the last value.* Thus, the reflection that must be performed conflicts with the true reflection of the coefficients.

This same observation is true of most wavelet transforms. The difference in the frame transform is the convolution with theta that is performed after the last level of forward transform.

For most wavelet transforms, the coefficients at the boundary of the highest resolution set of lowpass coefficients are used only by the inverse lowpass filter. Because of the way in which the filters and the reconstruction algorithm are designed, the reconstructed lowpass coefficients at the next lower resolution have the proper

Index	Lowpass	Highpass 1	Highpass 2
-8	0.0	0.0003	0.0
-7	0.0	0.0013	0.0
-6	0.0	0.0089	0.0026
-5	0.0	0.0293	0.0102
-4	0.0	0.0147	0.0175
-3	0.0	−0.0825	0.0187
-2	0.125	−0.1394	−0.0340
-1	0.5	−0.0739	−0.1881
0	0.75	0.4825	−0.1745
1	0.5	−0.0739	0.6950
2	0.125	−0.1394	−0.1745
3	0.0	−0.0825	−0.1881
4	0.0	0.0147	−0.0340
5	0.0	0.0293	0.0187
6	0.0	0.0089	0.0175
7	0.0	0.0013	0.0102
8	0.0	0.0003	0.0026

Table 3.10: Inverse Filter Values for a Spline Frame Filter

symmetry at the right boundary. For the frame transform, however, the theta convolution modifies these boundary values. Because the reflection symmetry at the right side of the array is inaccurate, the symmetry of the resulting coefficients will affect the operation of the reconstruction filter. This results in incorrect values at the right side of the array at each level of reconstruction. For our example, the three rightmost coefficients are in error after the convolution. This is because the theta filter, which has length seven, will not touch any reflected coefficients during the convolution if it is centred four or more coefficients from the end of the array.

It is worth noting that when $\theta = 1$, this problem does not exist. The problem arises only when the convolution with θ touches the values outside the array. When $\theta = 1$, this does not happen, and reconstruction proceeds correctly.

3.5.1 Solutions

Several solutions exist to this problem, as follows.

Changing the Reflection Strategy for Forward Transforms

Since this problem arises when reflection without repeating the last value is performed for the forward transform of the lowpass coefficients, one solution is to use the other type of reflection, namely to repeat the last value. This will avoid the problem, as the reflection now correctly captures the coefficients outside the boundary.

The difficulty with this solution, however, is that using the repeating reflection for the forward transform results in coefficients that do not have symmetry at the boundaries. The only way to achieve perfect reconstruction this way would be to store the extra values beyond the boundary. This is not desirable for compression, especially when the lowpass filters are long, as is the case with the Spline filter in our example.

Not Using Reflection

Another possibility is to avoid using reflection altogether. The best alternative is periodic extension, i.e. copying values from the other side of the array. This removes the problem, but, as was mentioned in Section 3.3, periodicity can lead to slightly skewed results, since one side of the image may be completely unrelated to the other side.

Extra Convolution

It should be noted that this problem does not occur when only one level of transform is performed. The reason is that after just one level of transform, there is no conflict in the reflection strategies, and thus the convolution with θ does not involve any incorrect coefficients. Thus, the reconstructed coefficients after this one level are completely correct.

The solution that results from this observation is to treat each level of transform separately. This means that for every level of reconstruction, convolution is performed before the reconstruction, and deconvolution is performed after the reconstruction, as if this one level of transform were the only one being performed. Since one level of transform works perfectly, we are guaranteed that after each level of reconstruction, the reconstructed coefficients are correct.

The net effect of the extra convolutions and deconvolutions is simply to fix the coefficients at the right side of the lowpass array. It is computationally expensive, but it allows reflection to be used without storing any extra values.

Keeping Convolution Coefficients

If a small amount of extra storage can be tolerated, another solution exists that saves a great deal of computation over the previous solution. Note that after each level of reconstruction, we should have the following stored in the lowpass array:

$$C^n * \theta$$

When the problem has not been fixed, we end up at each level with almost this set of data, but with some coefficients at the right side that are incorrect. If

these values can be calculated in advance, they could simply be substituted for the incorrect values after each level of reconstruction.

This precalculation can be done during the decomposition. It can be determined, based on the length of the θ filter, how many coefficients at the right side of the lowpass array will be affected by the convolution with θ after the last decomposition. Those coefficients can be calculated at each level, by performing the convolution with θ, and then stored. When the lowpass coefficients at that level have been reconstructed, the correct values cane be retrieved and substituted for the incorrect values.

3.6 Application of Frame Transform to Denoising

As was seen in section 3.2, implementing successful compression using the dual frame transform is a difficult problem, and beyond the scope of this thesis. In order to give some practical results, I decided to implement the dual frame transform within the context of denoising images.

Many approaches have been implemented for solving the denoising problem. For instance, Baraniuk describes in [Bar99] a tree-based system that makes use of linear programming. Given the desired maximum number n of wavelet coefficients to keep, this method determines the rooted subtree with only n nodes, of the wavelet pyramid that maximizes the square of its wavelet coefficients. The idea is to ignore the many small coefficients that result from the noise, as well as the few large ones, which are often isolated from other large coefficients. Selesnick and Sendur describe the results of applying this method on top of a tight wavelet frame transform in [Sel01].

For this purposes of this thesis, however, a simple approach was best. The focus of the thesis is not denoising, and thus the naive approach of hard thresholding was implemented. This approach involves setting all coefficients with magnitude below a pre-determined threshold to 0.

The results achieved from the sample program show the usefulness of the dual frame transform. Each pixel of each image was corrupted with additive white Gaussian noise, using a mean of 0 and a standard deviation of 5. Experimentation showed that the best threshold to use was 10, i.e. any wavelet coefficient with a magnitude below 10 was set to 0 prior to performing the inverse dual frame transforms.

The filter used for the denoising program is a linear spline framelet, with the filter coefficients shown in Tables 3.11 and 3.12.

Table 3.13 gives a summary of the numerical results achieved by the denoising program on a selection of sample images.

Figures 3.4, 3.5 and 3.6 show each of these images before and after the denoising.

It should be stressed that these results are quite basic, and do not address many of the crucial and difficult issues in denoising. They are intended to demonstrate

Index	Lowpass	Highpass 1	Highpass 2
-3	0.0	0.0	$-\sqrt{6}/24$
-2	0.0	0.0	$-2\sqrt{6}/24$
-1	0.25	-0.25	$6\sqrt{6}/24$
0	0.5	0.5	$-2\sqrt{6}/24$
1	0.25	-0.25	$-\sqrt{6}/24$
2	0.0	0.0	0.0
3	0.0	0.0	0.0

Table 3.11: Forward Filter Values for the Denoising Frame Filter

Index	Lowpass	Highpass 1	Highpass 2	θ
-3	0.0	0.0	$-\sqrt{6}/24$	0.0
-2	0.0	0.0	$-2\sqrt{6}/24$	0.0
-1	0.25	-0.25	$6\sqrt{6}/24$	$-1/6$
0	0.5	0.5	$-2\sqrt{6}/24$	4/3
1	0.25	-0.25	$-\sqrt{6}/24$	$-1/6$
2	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0

Table 3.12: Inverse and Theta Filter Values for the Denoising Frame Filter

Image	PSNR of noisy image	PSNR of denoised image
house	34.15	36.69
camera	34.11	36.61
bird	34.17	39.02

Table 3.13: Denoising results

only the applicability of the dual frame transform to practical applications.

3.7 Future Work

The work presented in this chapter is a small start towards achieving good compression results with the dual frame transform. A great deal of research must be performed in order to overcome the data redundancy issue and allow the excellent theoretical properties of the dual frame transform to show themselves. It is entirely possible that the SPIHT framework is not the best one for this particular transform. Other frameworks should also be investigated, or new ones created, to see which take the best advantage of the frame coefficients.

(a) (b)

Figure 3.4: The test image house (a) before denoising and (b) after

(a) (b)

Figure 3.5: The test image camera (a) before denoising and (b) after

(a) (b)

Figure 3.6: The test image bird (a) before denoising and (b) after

Chapter 4

Conclusion

The work presented in this thesis covers two different aspects of the image compression process. Hy-Q, presented in Chapter 2, introduces a novel way of mixing together the effects of two different quantization algorithms. The dual wavelet frame transform, presented in Chapter 3, is a new technique for mapping values from the spatial domain (image pixels) to the frequency domain (wavelet coefficients).

4.1 Summary of the Work

Hy-Q combines the quantization algorithms by first examining the properties of the image to be compressed. It employs a measure, based on the Kirsch edge detection algorithm and quadtrees, to see if each quadrant of the image is smooth or detailed. If all quadrants are smooth, then SPIHT's scalar quantization is used to compress the entire image. If all quadrants are detailed, then ModLVQ's vector quantization is used. If some quadrants are smooth while others are detailed, then a mixture of both algorithms is used.

The results of Hy-Q are encouraging. For uniformly smooth or uniformly detailed images, which Hy-Q decides to quantize with just one algorithm, Hy-Q provided better results than both SPIHT and ModLVQ in over 85% of the tests performed. For the images that feature both smooth quadrants and detailed ones, Hy-Q's results were at least equal to, and more often better than, both SPIHT and ModLVQ's results over 77% of the time.

For the dual wavelet frame transform, the goal was to implement the transform within an image compression framework. It was found that performing one level of transformation worked perfectly, whereas multiple levels of transformation resulted in incorrect data at one end of the reconstructed array. This problem can be solved in many different ways, some involving extra storage, others involving extra processing.

Because they employ multiple highpass filters, image compression using dual wavelet frames is challenging. The hope is that the freedom in design given by the extra highpass filters and the θ filter will allow the extra data to be accounted for

efficiently. Solving this problem was beyond the scope of this thesis, and thus a practical demonstration of the power of dual wavelet frames was given in the form of a simple denoising program.

4.2 Comparing the Projects

It is not possible at this time to make a fair comparison of the results achieved by these different techniques. While Hy-Q is a full algorithm that achieves good compression results, no work has yet been done to address the issue of the data redundancy in the dual wavelet frame transformations.

What links these two projects together is that they both offer significant avenues for further research into image compression, with the hopes of achieving state-of-the-art results. The main idea behind Hy-Q is easily transferable in principle to other quantization and image segmentation methods, to achieve superior results to those of either method alone. The dual wavelet frame transform offers excellent theoretical properties. This thesis has presented solutions to some of the practical implementation issues, paving the way for solutions to the data redundancy issues.

Bibliography

[ALI96] A. Averbuch, D. Lazar, and M. Israeli. Image compression using wavelet transform and multiresolution decomposition. *IEEE Transactions on Image Processing*, 5(1):4–15, January 1996.

[Bar99] R. Baraniuk. Optimal tree approximation with wavelets. In *SPIE Technical Conference on Wavelet Applications in Signal Processing VII*, page 3813, July 1999.

[Can86] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, November 1986.

[Can98] E. Candes. *Ridgelets: theory and applications.* PhD thesis, Stanford University, 1998.

[Cas96] K. R. Castleman. *Digital Image Processing.* Prentice Hall, 1996.

[CD99] E. Candes and D. Donoho. Ridgelets: a key to higher-dimensional intermittency. *Phil. Trans. R. Soc. Lond. A.*, pages 2495–2509, 1999.

[DH00] I. Daubechies and B. Han. Pairs of dual wavelet frames from any two refinable functions. *Constructive Approximation*, submitted, 2000.

[DV00] M. Do and M. Vetterli. Orthonormal finite ridgelet transform for image compression. In *ICIP00*, page TA10.02, 2000.

[FA97] J. E. Fowler and S. C. Ahalt. Adaptive vector quantization using generalized threshold replenishment. In J. A. Storer and M. Cohn, editors, *Proceedings of the IEEE Data Compression Conference*, pages 317–326. IEEE Computer Society Press, March 1997.

[Hea97] Michael T. Heath. *Scientific Computing - An Introductory Survey.* McGraw Hill, 1997.

[HGS97] R. Hamzaoui, B. Ganz, and D. Saupe. Quadtree based variable rate oriented mean shape-gain vector quantization. In J. A. Storer and M. Cohn, editors, *Proceedings of the IEEE Data Compression Conference.* IEEE Computer Society Press, March 1997.

[HM96] P. J. Hahn and V. J. Matthews. Distortion-limited vector quantization. In *Proceedings of the IEEE Data Compression Conference*, pages 340–348. IEEE Computer Society Press, 1996.

[JLN99] A. Jarvi, J. Lehtinen, and O. Nevalainen. Variable quality image compression system based on spiht. *Signal Processing: Image Communications*, 14:683–696, 1999.

[Kni96] J. Knipe. Improved spatial and transform domain compression techniques. Master's thesis, University of Alberta, August 1996.

[MGBB00] M. W. Marcellin, M. J. Gormish, A. Bilgin, and M. P. Boliek. An overview of jpeg-2000. In *Proceedings of the IEEE Data Compression Conference*, pages 523–541, 2000.

[RH99] M. Ramos and S. Hemami. Activity selective spiht coding. *SPIE Conf. on Visual Communications and Image Processing*, 3653:315–326, January 1999.

[SdSG94] D.G. Sampson, E.A.B. da Silva, and M. Ghanbari. Wavelet transform image coding using lattice vector quantization. *Electronics Letters*, 30(18):1477–1478, 1994.

[Sel01] I.W. Selesnick. Smooth wavelet tight frames with zero moments. *Appl. Comput. Harmon. Anal.*, 10(2):163–181, 2001.

[SF94] E. Shusterman and Meir Feder. Image compression via improved quadtree decomposition algorithms. *IEEE Transactions on Image Processing*, 3(2):207–216, March 1994.

[Sha93] J.M. Shapiro. Embedded image coding using zerotrees of wavelet coefficients. *IEEE Transactions on Signal Processing*, 41(12):3445–3462, 1993.

[SN97] G. Strang and T. Nguyen. *Wavelets and Filter Banks.* Wellesley-Cambridge Press, 1997.

[SP96] A. Said and W.A. Pearlman. A new fast and efficient image codec based on set partitioning in hierarchical trees. *IEEE Transactions on Circuits and Systems for Video Technology*, 6:243–250, 1996. Available at http://www.cipr.rpi.edu/research/SPIHT/EW_Code/csvt96_sp.ps.gz.

[Str90] P. Strobach. Tree-structured scene adaptive coder. *IEEE Trans. Commun.*, 38(4):477–486, 1990.

[Tha96] Nguyen T. Thao. A hybrid fractal-DCT coding scheme for image compression. In *Proceedings ICIP-96 (IEEE International Conference on Image Processing)*, volume I, pages 169–172, Lausanne, Switzerland, 1996.

[Uhl96] A. Uhl. Adaptive wavelete image block coding. In H. H. Szu, editor, *Wavelet Applications III*, volume 2762, pages 127–135. SPIE Proceedings, April 1996.

[WF95] X. Wu and Y. Fang. A segmentation-based predictive multiresolution image coder. *IEEE Transactions on Image Processing*, 4(1):34–47, January 1995.

[ZASB95] A. Zandi, J. D. Allen, E. L. Schwartz, and M. Bolick. Crew: Compression with reversible embedded wavelets. In *Proceedings of the IEEE Data Compression Conference*, pages 212–221, March 1995.